Heriot-Watt University

Masters Thesis

Investigate the use of Discrete Dynamical systems within recurrent neural networks for chaotic time series predictions

Author:
Matteo CISNEROS

Supervisor:
Dr. Micheal LONES

A thesis submitted in fulfilment of the requirements for the degree of MSc.

in the

School of Mathematical and Computer Sciences

August 2023

Declaration of Authorship

I, Matteo CISNEROS, declare that this thesis titled, 'Investigate the use of Discrete Dynamical systems within recurrent neural networks for chaotic time series predictions' and the work presented in it is my own. I confirm that this work submitted for assessment is my own and is expressed in my own words. Any uses made within it of the works of other authors in any form (e.g., ideas, equations, figures, text, tables, programs) are properly acknowledged at any point of their use. A list of the references employed is included.

Signed: MATTEO CISNEROS

Date: 04/16/2023

Abstract

Training a recurrent neural network (RNN) can be hard, time-consuming, and power-consuming. In the early 2000s, a new model inspired by RNNs has been proposed as a solution to those problems, reservoir computing (RC). The concept of RC appeared with two algorithms: Echo state network (ESN) and Liquid state machine (LST). The main feature is that RC reduces considerably the training time, it is composed of two parts, the reservoir randomly initialized that will not be trained, and a readout layer to process the reservoir's output that will be trained. This paper will investigate different types of discrete dynamical systems (DDS) as a replacement for the randomly initialize reservoir. Those reservoirs will be Binary Elementary Cellular Automata (ECA), three states ECA and Coupled Map Lattices (CMLs). Those reservoirs will then be tested for chaotic time series prediction with different configurations. During this experiment, we will use the edge of chaos rules for our Binary ECA to ensure Turing completeness and so, the best computation. We found that overall the ESN has better accuracy and is faster but is less stable, discrete dynamical systems (DDS) have a smaller radius for accuracy and are more robust to larger datasets.

Acknowledgements

I would like to thank my supervisor Michael Lones for his help and support during this project.

Contents

List of Figures

List of Tables

Abbreviations

CA	**C**ellular **A**utomata
GoL	**G**ame **o**f **L**ife
ECA	**E**lementary **C**ellular **A**utomata
RC	**R**eservoir **C**omputing
ReCA	**Re**servoir **C**ellular **A**utomata
ESN	**E**cho **S**tate **N**etwork
LSM	**L**iquid **S**tate **M**achine
NN	**N**etwork **N**eural
RNN	**R**ecurrent **N**eural **N**etwork
FFNN	**F**eed **F**orward **N**eural **N**etwork
CML	**M**ap **C**ouples **L**attices
DDS	**D**iscret **D**ynamical **S**system

Chapter 1

Introduction

The aim of this project is to show the performance of a cellular automata based model for time series forecasting and compare them to other types of reservoirs.

1.1 Motivation

Today cellular automata has already a long history with various applications and papers, and it is still being researched by various teams across the world. On the other hand reservoir computing is a fairly new concept dated in the early 2000s that shows good promises for various applications in sequential data processing. Despite the growing interest in RC, there are not a lot of papers on time series prediction using cellular automata reservoirs. To work on a project that merges together a well-known concept as cellular automata and a fairly new one as reservoir computing to implement a powerful tool able to resolve hard problems is a great motivation.

1.2 Objective

The project aims to demonstrate the feasibility of the use of a cellular automaton as a reservoir for reservoir computing, but also the efficiency of this type of reservoir for time series forecasting. In this project, we will implement a cellular automata reservoir and test it for chaos time series prediction and measure its accuracy, efficiency, and scalability.

1.3 Expected Results

The expected results will demonstrate the effectiveness of using cellular automata as a reservoir for time series prediction. We expected to see the cellular automata reservoir outperformed other types of reservoirs as the Liquid state machine or the Echo state network in both accuracy and computational efficiency.

Overall this project aimed to find the best parameters for a cellular automata reservoir and to show the feasibility of such a model for time series forecasting.

Chapter 2

Literature Review

2.1 Cellular Automata

The concept of cellular automata (CA) come from the work of Stanislaw Ulam about the growth of crystals and the work of his colleague John von Neumann about the problem of self-replicating systems in the 40s, the first idea of "cellular automata" come from Stephen Wolfram in the 80s, the birth of cellular automata is explained in this book [Schiff, 2011]. CA has since been applied to an extensive range of fields including computer science, biology, and physics. CA are mathematical models that simulate complex systems with simple or complex rules to update the state of cells in an environment usually a grid.

The particularity of CA is its ability to exhibit emergent behavior, where complex patterns and structures emerge from simple interactions between the cells in the grid.

The first work on CA where in one dimension called elementary cellular automaton (ECA). In this configuration cells in the grid could only have two states usually 1 or 0. Today we know four different classes of CA rules for ECA;
class I: Rules define a system where patterns evolve quickly into a stable, homogeneous state [Ilachinski, 2001],
class II: Rules define a system where patterns evolve quickly into stable or oscillating structures [Ilachinski, 2001],
class III: Rules define a system where patterns evolve in a pseudo-random or chaotic manner. Any stable structures that appear are quickly destroyed by the surrounding

noise [Ilachinski, 2001],
class IV: Rules define a system in a transition phase between chaos and order, edge of chaos [Ilachinski, 2001]. These are the most interesting rules as they ensure Turing completeness, they will be detailed in section 2.1.1.
Rendell [2002] proved in 2002 that Conway's Game of Life (GoL) could be used to build a Turing machine, this one is a set of rules devised by John Horton Conway in 1970 [Gardner, 1970] and is today used in various applications, simulations, reservoir computing, and more.
CA is a powerful tool to study many aspects including pattern formation, optimization for a complex system, cryptography, or social simulations.

However, CA like every other system has its limitations. Indeed, the accuracy is sensible to the prior settings of the automation and can be expensive in a computing way as the larger the grid, the more the computations.

Nonetheless, CA is a powerful tool used in many fields for many purposes. It is today an active area of research, like cellular automata for reservoirs.

The types of neighbors and boundary conditions will be covered in respective sections 2.1.2 and 2.1.3.

2.1.1 Class IV rules

We said in section 2.1 that the class IV rule for CA was one of the most interesting ones because of its state at the edge of chaos. This means that those rules exhibit complex, apparently random behavior, with patterns that are difficult to predict or understand, this behavior helps simulate turbulence and randomness in natural systems such as weather or economic systems, stock market prices and wind speed are just examples. Stephen Wolfram investigated the behavior of such rules, he found that type IV rules exhibited a wide range of behaviors, from simple oscillations to complex and seemingly random patterns, and this will make them useful tools for modeling complex systems.
Two examples of such rules are rule 54 and rule 110 and they can be extended for two-dimensional CA, in this case, we use John von Neumann's neighbour 2.2 and two states, "on" and "off":

- Rule 54, can support glider-like structures and other emergent patterns and is defined as follow.

  A cell that is currently "off" and has exactly two of its eight neighbours "on" will take the "on" state.

  A cell that is currently "on" and has at least one of its eight neighbours "on" will take the "off" state.

  Otherwise, the cell remains in its current state.

- Rule 110, can support universal computation and is defined as follow.

  A cell that is currently "on" and has its eight neighbours "off", will take the "off" state.

  A cell that is currently "off" and has exactly one of its eight neighbours "on" will take the "on" state.

  Otherwise, the cell remains in its current state.

2.1.2 Neighbourhoods

One of the most important parameters for a CA is the type of neighbors. The neighbourhood of a cell is the nearby, usually adjacent, cells and sometimes the cell itself. There are a lot of types of neighbours and each of them will affect the system differently. Neighbour is the manifestation of the rules in the grid, the whole behavior of the system will be determined by this neighbourhood and the rule applied to it. The most known types of neighbours are John von Neumann's neighbour and Moore's neighbour [Crooks, 2017] as shown for (a)-(b) in figure 2.1 however there is no theoretical limit to the size and shape of a CA neighbourhood other than the size of the grid and the computational power of the machine running the model.

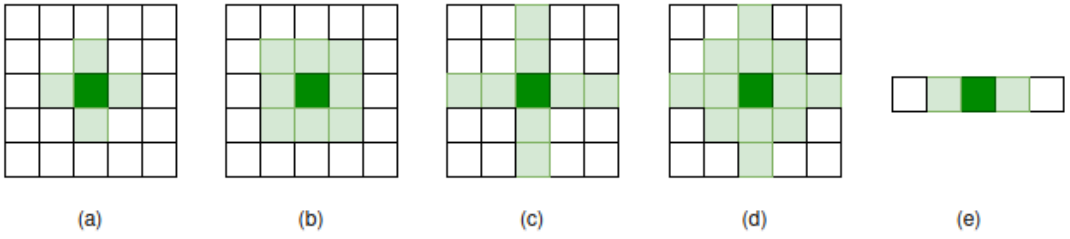

FIGURE 2.1: Neighbourhood - (a)John von Neumann's neighbour - (b)Moore's neighbour - (c)Extended von Neumann's neighbour - (d)Moore von Neumann's neighbour

Other well-known neighbours are Extended von Neumann or Moore von Neumann

(MvonN) which is a combination of (a) and (b) in figure 2.1. In elementary cellular automata, there is not a lot of different type of neighbouring, the one usually used is the neighbour at the right, left and the cell itself as shown in (e) in figure 2.1.
However, the greater the number of neighbours, the more complex computation is needed to determine the next state of cells in the grid.

2.1.3 Boundary Conditions

Another important parameter for CA is the type of boundary conditions, this applies to the neighbor of the cell located near the border of the grid (if this one is finite). As we said there are multiple ways to handle this event; periodic: the cell will take the state of the one at the exact opposite of the grid, reflecting: the cell will take the state of the one at the exact opposite of the current cell as investigated in [Uguz and Redjepov, 2021], adiabatic: the cell will take the state of the current one, as investigated in [Uguz and Redjepov, 2021], fixed: the cell will take state set before as a parameter. All those boundary conditions are shown in figure 2.2 with a Moore neighborhood.

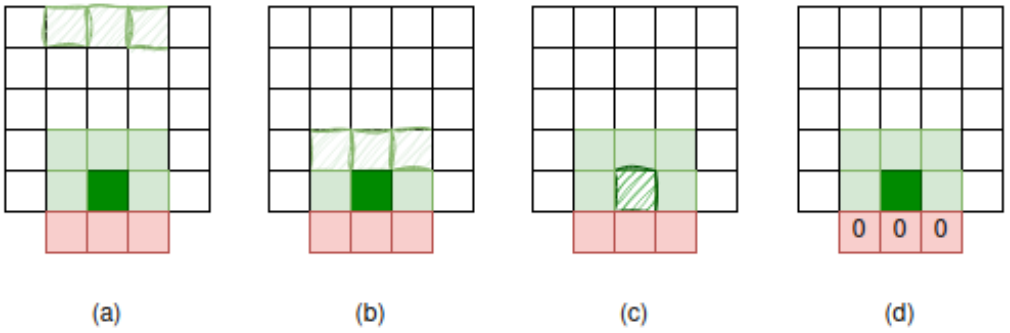

FIGURE 2.2: Boundary conditions - (a)Periodic - (b)Reflecting - (c)Adiabatic - (d)Fixed state 0

2.1.4 More CA

We said in section 2.1 that today CA was used for a lot of purposes, including cancer growth simulation [Qi et al., 1993] and the evolution of an elementary CA governed by quantum mechanics [Grössing and Zeilinger, 1988]. Those applications of CA show that

they are very versatile and can be applied to physics concepts and more. This proves that CA is computationally very interesting particularly if you use Turing complete rule for elementary or 2D CA, this allows us to simulate complex patterns and random behaviors. More CA work related to reservoir computing will be discussed in section 2.7.

2.2 Coupled Map Lattices

The concept of Coupled Map Lattices (CMLs) was first introduced by Grebogi et al. [1982] in 1982, CML are discrete dynamical systems with discrete time and space and continuous state variables. The paper demonstrates that simple coupled maps like the logistic map could exhibit complex spatiotemporal patterns and chaos when organized in a lattice structure. Later in 1992, a research paper written by Kaneko [1992] introduced a new equation to study spatiotemporal chaos through local nonlinear dynamics. The equation and its explanation are detailed here 5.1.3. Today CMLs are used in various more field like biology, ecology, computer science, and more, this is why CMLs is today a topic of extensive research and investigation. This concept will be used within a recurrent neural network and compared to other types of reservoirs. Figure 2.3 presents an example of single CMLs dynamics.

2.3 Reccurent Neural Network

A recurrent neural network (RNN) is a type of artificial neural network based on Rumelhart et al. [1986]'s work from 1986, it is designed to process sequential data for speech recognition or time series prediction. The special characteristic of this type of neural network is that it is composed of loops that allow them to process and remember information from previous steps, this is represented in figure 2.4. An RNN process the input data based on the current state of the network and the input. From this, the network can create a type of memory inside itself that can apprehend deeper patterns.
In conclusion, RNN is a powerful tool to make a temporal relationship between inputs and so process large sequential data. This tool is valuable when solving complex problems in various fields including physics, biology, and more.

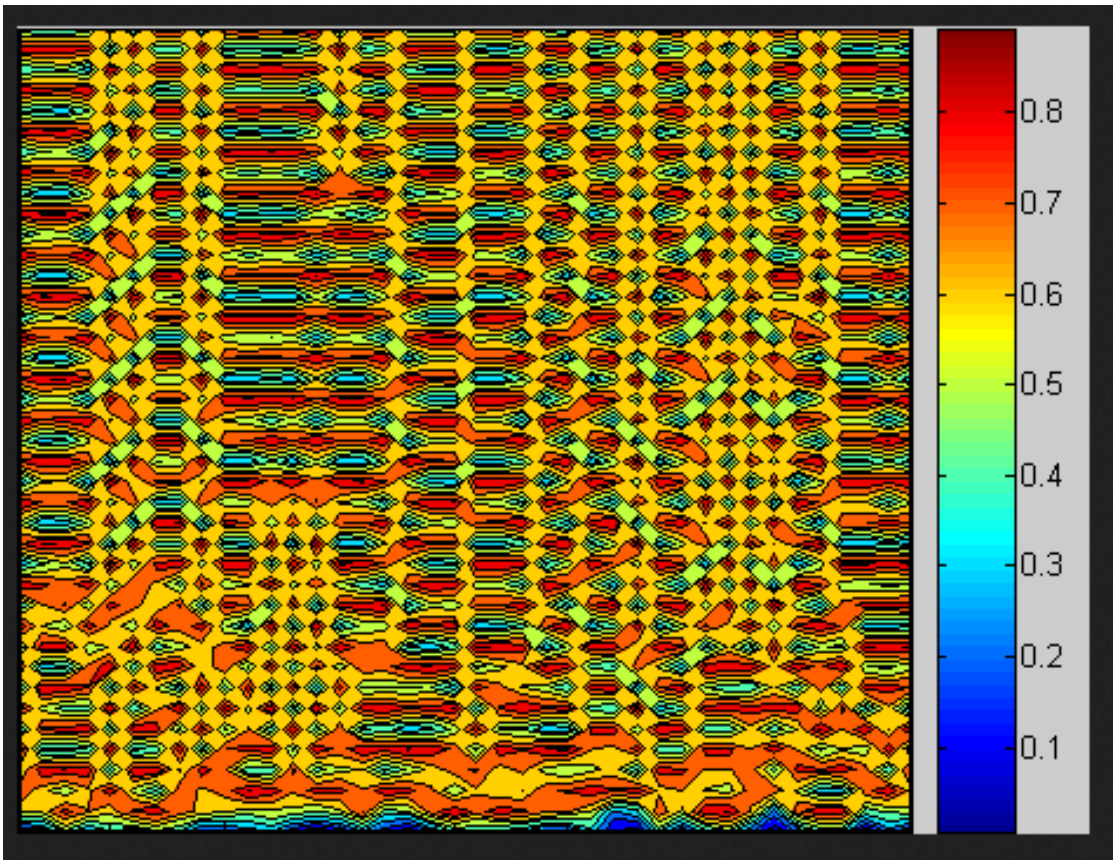

FIGURE 2.3: Single neighbor CMLs dynamics over forty iterations

2.4 Reservoir

Reservoir computing (RC) is a fairly new concept dating from the 2000s traced back to the "Echo state network" and "Liquid state machine", both will be detailed in respective sections 2.4.2 and 2.4.3.

The reservoir is like a black box filled with neurons that acts as a dynamic memory system that uses past information to precise its predictions, it is not trained on the specific task. Instead, the reservoir is randomly initialized and remains fixed, each neuron's connection is also randomly initialized. In addition, some neurons have a connection to themselves as they will use their output in the next process. After the reservoir, an output layer called "Readout" is updated during the training and will process the information coming out of the reservoir, the concept of RC is shown in figure 2.5. The readout is usually a neural network here it will be a simple Feedforward neural network covered in section 2.4.1.

There are a lot of advantages to using RC one of them is that it can be applied to multiple problems including classification, speech recognition, and time series prediction. Another

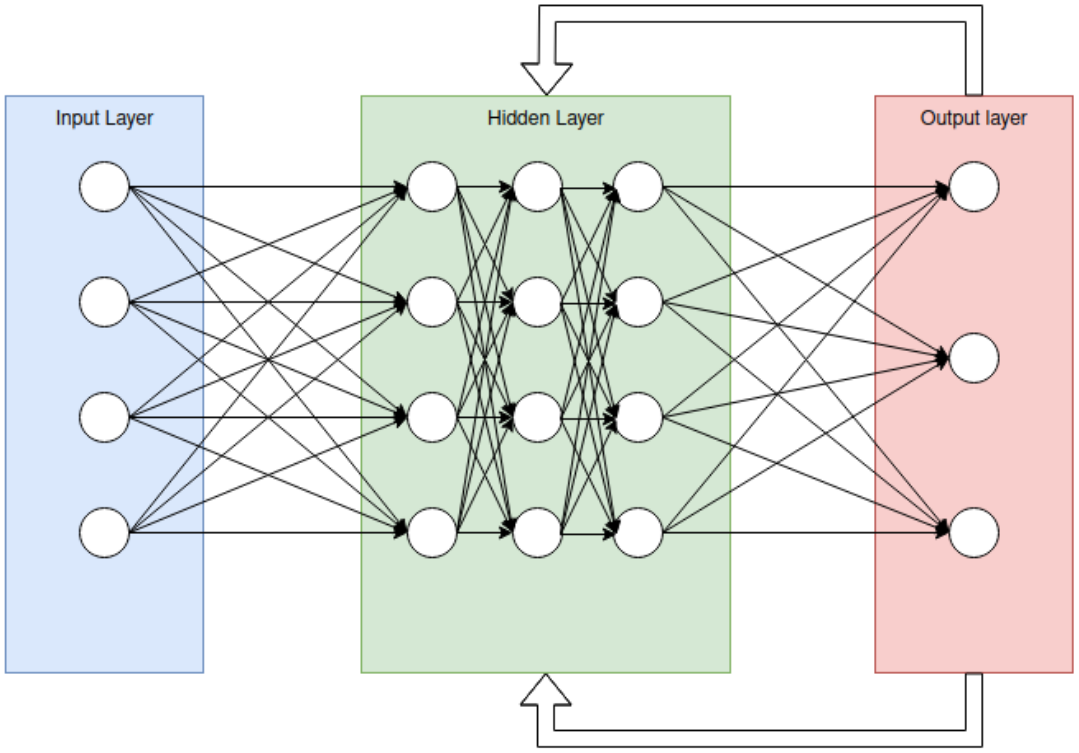

FIGURE 2.4: Recurrent Neural Network

is that it can handle a large amount of data because the reservoir does not need training we only have to train the readout.

In conclusion, RC is a powerful machine-learning method that is applied in many fields, for complex problems.

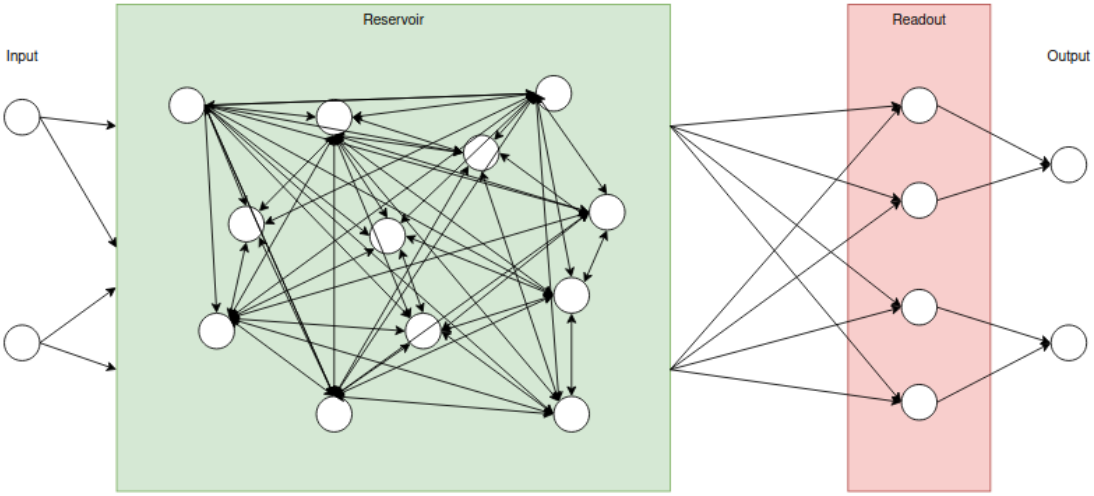

FIGURE 2.5: Reservoir Computing

2.4.1 Feedforward Neural Networks

Feedforward neural network (FFNN) is a type of neural network that dates back to 1958 with the work of Frank Rosenblatt about a model of a single-layer neural network called the perceptron, [Rosenblatt, 1958]. FFNN is one of the first and simplest neural networks implemented today. As its other name multi-layer perceptron implied FFNN is composed of a fixed number of layers, an input layer, an output layer, and one or more hidden layers, each layer is composed of a fixed number of perceptron that can be different for each layer, the settings of this neural network will depend on the problem we try to resolve. A perceptron or McCulloch-Pitts neuron discovered in 1943 with [McCulloch and Pitts, 1943], is an algorithm composed of an activation function that maps inputs into an output. In an FFNN each neuron of each layer is linked to every neuron of the next layer.

What is called "feedforward" means that data is passing in the model from one way to another (input layer to output layer). In the figure 2.8 from right to left. There are multiple ways to train an FFNN but the most common is backpropagation, where we use the output of each perceptron to update its settings and the accuracy (in the figure 2.8 from right to left).

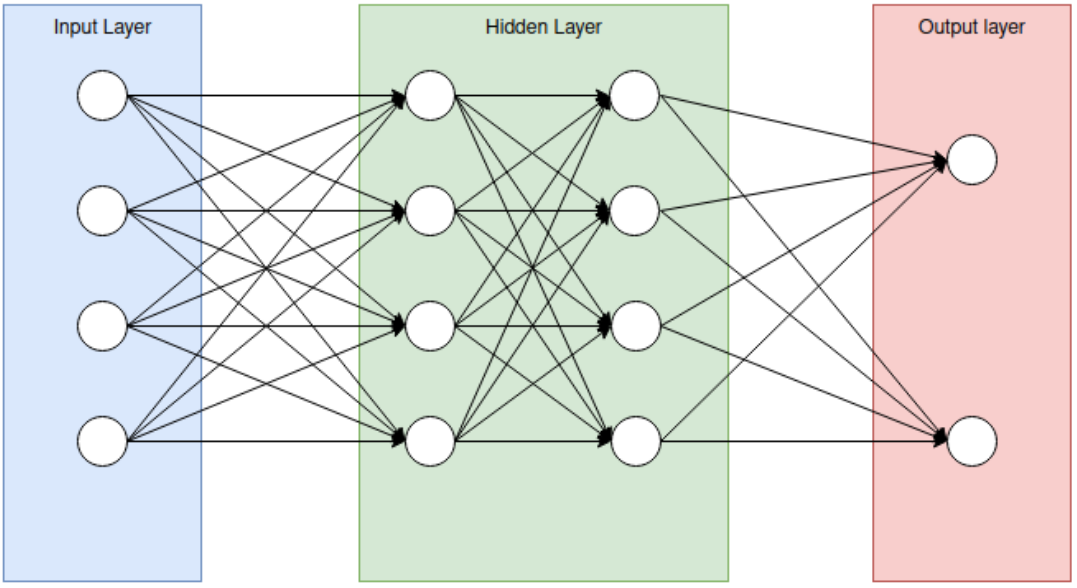

FIGURE 2.6: Feedforward Neural Network

$$\|y - Xw\|_2^2 + \alpha\|w\|_2^2$$

FIGURE 2.7: Here $\|w\|_2^2$ denotes the L2 norm or Euclidean norm, which is a way of measuring the magnitude or length of a vector.

2.4.1.1 Ridge Regression

The Readout layer will not be implemented from scratch, we will use a well-known library called "sklearn linear model". In this library, a number of networks exist, the one we are interested in is the Ridge Regression for ESN, CA, and CML. In this model, we try to minimize a compact version of the loss function of the Ridge regression denoted here 2.7.

In this equation, y represents the target variable we want to predict, X is the matrix of input features and w is the weights vector. The last parameter alpha is used to regulate the behavior of the regularization by penalizing large coefficients, the smaller the alpha, the less penalized the big coefficient. This has for purpose to avoid overfitting and encourage the model to have smaller coefficient values.

In conclusion, Ridge regression seems to be a better choice than a Linear regression for various reasons: a stronger robustness to outlinear and to overfitting thanks to the alpha parameter and, it is also better when handling multicollinearity, which is a situation where two or more independent features are highly correlated. The Ridge regression introduces a regularization term denoted $||w||2^2$, it will force the equation to compute smaller coefficient values and so minimize the impact of features that are highly correlated.

2.4.2 Echo state network

The echo state network (ESN) is a type of RNN researched in 2004 by Jaeger and Haas [2004] used for machine learning. In an ESN, the input data is fed into the reservoir, which processes the data and generates a high-dimensional state vector. The output layer of the network is then trained to generate predictions based on the current state of the reservoir. An example of a very interesting task for ESN is reinforcement learning as investigated here [Szita et al., 2006]. The memory part of the ESN is needed to improve the reinforcement part of reinforcement learning.

2.4.3 Liquid state machine

The liquid state machine (LST) is a type of RNN first introduced in 2002 by Wolfgang Maass and Thomas Natschläger, [Maass, 2011], the main concept of LSM was for computational neural science. The main particularity of LST is that the connection between the neurons in the reservoir is constantly changing to make the reservoir learn about the data. The main difference between LST and ESN is that LST uses differential equations to process sequential data whereas ESN updates the state of the network at discrete time steps that allow a simpler implementation.

2.4.4 LST/ESN/RNN

LST and ESN are different types of reservoir computing inspired by RNN, and each of these models has its strengths and weaknesses:

- Architecture: the main difference here is between RNN and LSM/ESN, indeed, where RNN is a simple feedforward with some additional connection of a neuron with itself, ESN and LSM are random initialize recurrent network where the output of each node is sent back to the reservoir as input.
- Training: RNN is typically trained with backpropagation through time (BPTT) algorithm where we send the error backward in the system to update the weights. On the other hand, LSM and ESN do not need training for the reservoir part, only the readout will be trained.
- Performance: the performance of these models varies with the problem they try to solve, RNN is often used in language processing and sequence modeling, LST, on the other hand, is used for speech recognition and ESN is commonly used for time series prediction and classification.

2.5 Cellular automata for reservoir computing

This section is the union of section 2.1 and section 2.4. Professor Ozgur Yilmaz researched cellular automata reservoir (ReCA) in 2017 [Yilmaz, 2014]. The main concept of ReCA is that the reservoir is not a neural network anymore, it is replaced by a cellular

automaton where the data is initialized in the CA grid and the final state of the CA is used as input in the readout layer to make predictions.

The advantages of using such a model are that it needs fewer computing elements and the behavior of the reservoir can easily be altered, thus ReCA can be applied to a wide range of problems. We said in section 2.1 and section 2.1.1 that with type IV rules a CA can simulate complex dynamics, sometimes random with a high level of computation while staying easily adjustable, these strengths allow ReCA to have sometimes better accuracy than other types of reservoir computing model.

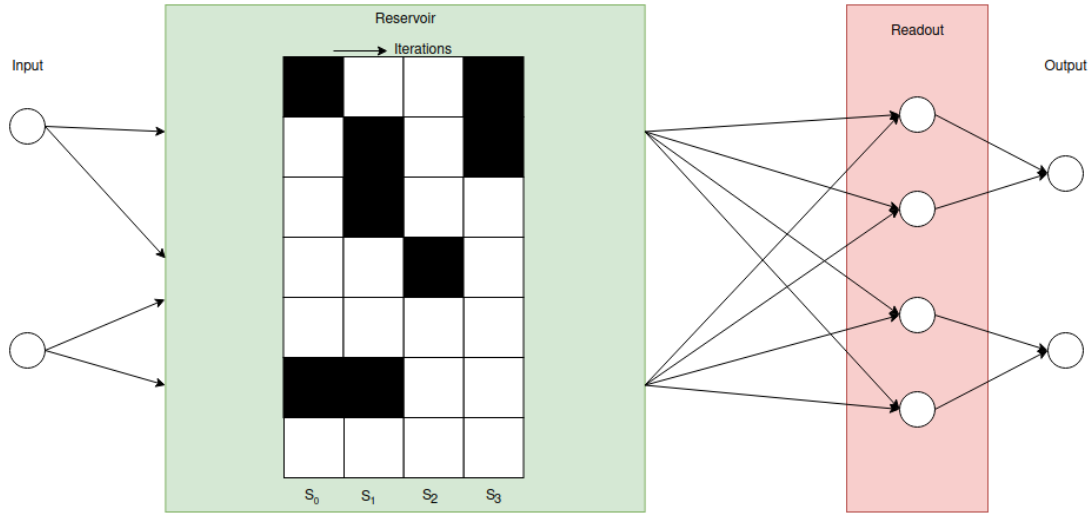

FIGURE 2.8: Cellular automata for reservoir computing

2.6 Time Series Prediction and dataset

Time series prediction is a common task used in machine learning, it consists in forecasting the future values of a time series, a sequential set of data measured over time at regular intervals. On the other hand, the case of chaotic time series prediction consists in forecasting the future behavior of a chaotic dynamic system. The main characteristic of a chaotic system is its sensitivity to its initial conditions, a little change in these conditions will change the whole behavior of the system. In this project we will use chaotic time series prediction to test our ReCA, to do this we will use three different datasets that describe such chaotic behaviors (these three datasets are selected from Kaggle, and their legal and ethical issues are discussed in section 7):

- Sunspot is a dataset containing data about solar eruptions collected by the Solar Influences Data Analysis Center in Belgium. This dataset is a great example of a natural phenomenon that is hard to predict. This dataset describes a difficult natural phenomenon it is so not that big with 3265 records from 1749 to 2017.

- DIJA company stock price is a dataset about the DIJA company and its price on the stock market, these data are collected from the Yahoo website. This example is about a human phenomenon impacted by countless numbers of people and conditions, that make it hard to predict. This dataset is the smallest one, it has only 1989 records, we will have to maximize each and any one of them.

- Wind speed this last dataset is of the same kind as the one about sunspot as wind is also a difficult phenomenon to predict. This is our biggest dataset but compare to today's big datasets this one is small with 6574 records.

2.7 Related work

Today ReCA is an important subject of research due to its qualities cited in section 2.4. However, today's research goes further into the concept with the coming examples.

2.7.1 Deep cellular automate reservoir

A research group in Oslo, Norway led by Stefano Nichel is working on a deep cellular automata reservoir (DReCA) as explained in this paper [Nichele and Molund, 2017*a*]. In a deep reservoir system, the output of one reservoir is sent to the other since we do not have to train the hidden layers in the reservoir, where the reservoirs are stacked in a deep architecture. This type of architecture is inspired by deep neural networks where adding layers helps increase the representation and accuracy. The main argument for doing so is that one reservoir's error can be fixed by the next one. In [Nichele and Molund, 2017*a*] the five-bit task memory is used as a benchmark to test the deep architecture (see the paper for the five-bit task memory explanation). The results show that the next reservoir does improve the results of the last one to a certain degree. This work is just the beginning for DReCA. This new time of reservoir has only been evaluated on one problem thus, it would need to be evaluated on different problems at a bigger scale to

prove its efficiency. In addition, this work mostly focuses on one metric, accuracy, it will be a good idea to check other metrics such as computation time or scalability.

2.7.2 Deep reservoir computing

Before the deep cellular automata reservoir, a deep echo state network model has been proposed in 2018 by Gallicchio et al. [2018]. From this work, a variety of deep reservoir computing models have been proposed and tested as the work of a team in Prague by Cisneros et al. [2022]. This work is using different types of reservoirs and tested them on a benchmark of increasingly difficult tasks. The results show that reservoir-based algorithms perform better than fully supervised algorithms. This shows that ReCA performs well on multiple tasks and is very versatile. In this work, we take a stronger look at different metrics and different level of strength of a problem, this shows how the reservoir evolve on different task for some too hard for the human brain.

2.8 Summary

Overall, this chapter defined the necessary part to understand this project, such as cellular automata, reservoir computing, and time series prediction. This literature review shows that reservoir computing is still an unknown field where there is still a lot to do. On the other hand, cellular automaton is a well-known model but it is still today widely used and has sometimes better performance than neural networks, especially in the case of reservoir computing.

In this project, we will use a cellular automaton as a reservoir for reservoir computing and then investigate the performance and scalability of this model on different datasets for time series prediction.

Chapter 3

Requirements Analysis

This project aimed to investigate the use of cellular automata as a reservoir for time series prediction. Requirements analysis is an important step in a project, this will help understand and/or prevent delays, understand the scope, and the risks. In order to work in the perfect conditions some requirements are needed and are outlined in this section.

3.1 Requirements Table

The requirements table is detailed here 3.1. This table has been made with the MoSCoW strategy Clegg and Barker [1994], where there are four different types of priority: Must, Should, Could, and Won't. This strategy helps define the priorities and differentiate the important parts from the less important ones.

Number	Requirement	Priority
1	Investigate a range of CA rule sets and neighbourhoods	Must
2	Investigate a range of Readout	Must
3	Evaluate the reservoir on each dataset	Must
4	Evaluate the results with different types of reservoir	Should
5	Find best CA settings with evolutionary algorithm	Could
6	Evaluate reservoir performance with different metrics	Could
7	Evaluate reservoir on a different problem	Won't

Table 3.1: Requirement table

3.2 Implementation

This project's implementation requires a number of different features, which are defined here:

- Implement the CA grid, rule 54 and 110.
- Implement readout to make predictions from the reservoir process. For this part, we will use the Python library called scikit-learn to create a simple feedforward neural network.
- Implement a preprocess of the three different datasets for CA.
- Train the model with our data.
- Evaluate the reservoir against other types of reservoirs. We will use the Python library ReservoirPy as it allows us to create an echo state network model very fast.

3.3 Risk assessment

In a project, there will always be risks, therefore it is important to identify them as soon as possible and if possible find a way to mitigate them.
The most common is the risk of being behind schedule, to minimize the chance of this becoming a problem we did a Gantt chart 3.1, which is a project planning tool often used by teams for work planning.
Other possible risks are defined here 3.2 with more information.

Risk	Impact	Likelihood	Mitigation
Behind Schedule	Medium	Medium	Change scope, focus on important tasks
Illness	Medium	Medium	Ty to do as much as possible, change the scope, focus on important tasks
Hardware failure	High	Low	Fix computer if possible or replace it
Bad implementation	Low	Low	Take time do understand the mistake, restart from scratch if necessary
Loss of data	High	Low	Make regular backups on GitHub for example

TABLE 3.2: Risk assessments

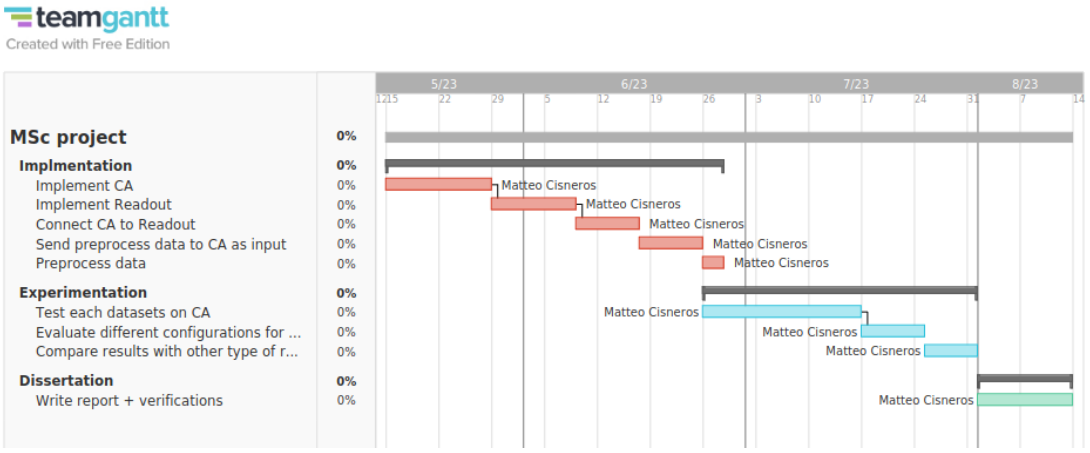

FIGURE 3.1: Gantt chart

3.4 Evaluation

For this experimentation part of this project, we will have two different steps. First, we will test our reservoir on the three different datasets 2.5 and investigate different configurations of the CA to find the parameters. Then, we will compare our results with different types of reservoirs on the same dataset. Our evaluation will be based on different metrics: accuracy, computation time, and scalability as our datasets are different in the data and size of this data. Each reservoir and configuration will be executed at least 20 times to get a good approximation. As described in section 2.6, our

datasets are fairly small so we will have to maximize their use with in this case the use of cross-validation as an evaluation method.

Chapter 4

Professional, Legal, Ethical, and Social Issues

This chapter discusses the professional, legal, ethical, and social issues that applied to this project.

4.1 Professional Issues

This project is the final mark for an MSc degree in data science at Heriot-Watt University Edinburgh. Every code, tool, and literature used are documented according to their license and cited appropriately.

4.2 Legal Issues

All the data used in this project are free and publicly accessible on the Kaggle website. The three datasets used do not imply any legal issues.
The project is made using the Python programming language owned by the Python Software Foundation (PSF), which uses the PSF license agreement, allowing free use of Python.

4.3 Ethical Issues

The experimentation in this project implies data with no personal data, and no human tests will be required, only the author and a computer.

4.4 Social Issues

Machine learning is a popular tool used today for many applications by a large number of companies around the world. The question of social issues can be asked with today's large quantities of personal data across the internet, the author will not use any type of personal data for this project. At the time of writing the author is also unaware of any malicious examples of CA or RC in the literature.

Chapter 5

Implementation

5.1 Dynamical systems

A dynamical system is a framework used to describe the behavior of a system over time, it uses mathematical expression or a set of rules to simulate the evolution of various phenomena across various fields. Two different types of dynamical systems are used in this dissertation, Cellular Automata and Coupled Map Lattices.

5.1.1 Cellular Automata

In this section, the general framework of the system is described in Figure 5.2. Elementary cellular automata is used as a reservoir, each cell has three neighbors including itself, and can take one of two states (here 0 or 1).

5.1.1.1 Encoder

The first step is to encode the incoming data into binary values in order to map them in our CA, a threshold with value 0.5 is used, all continuous values under this threshold are set to 0 and the remaining to 1.

The size of the grid is crucial for the success of the systems, if the grid is too small the reservoir will not be able to evolve correctly in the contrary if it is too big there will be too much noise and the accuracy will diminish. In this dissertation, the size of the grid

will be defined as followed: $I * R * C$, [Babson and Teuscher, 2019].
The number of iterations denoted by I refers to the number of times we will update the system.
Parameter R refers to a mechanism called replication, the input data is mapped X number of times in the grid in a random way where with each replication the new states of the chosen cells are overwritten in a grid of size $replication * iteration$.
The C-parameter (or C) [Nichele and Molund, 2017*b*], refers to the number of times the encoder will create a CA grid and map the input, at the end, all the different CA grid will be concatenated to one another in order to create one CA where the input is recalled multiple times.
The reason behind this computation is to capture a balance between the complexity of the reservoir's dynamics, the duration of the evolution of the input and the degree to which the data is amplified. By tuning this parameters we can modify the reservoir's characteristics to match the requirements of the ongoing task.
The encoder also has another task called padding, this refers to the method of adding elements of no information in this case, zeros to fill the remaining cells of the grid where no data has been mapped to add noise in the system.

5.1.1.2 Execution

After encoding, we apply a rule to the automaton that will enable nonlinear evolution, this way the automaton will be able to exhibit a wide range of behavior and patterns that cannot be easily achieved with a linear environment and rules, [Nichele and Molund, 2017*b*].

The system will be updated following the iteration parameter, each I a rule or a combination of rules will update the state of each cell, see 5.1.1.3. To process sequential data and get the best accuracy possible we need to develop a kind of "long short-term memory" in the system two mechanisms are used to achieve this:

- *Time transition function*

  In order to allow the system to remember previous input and so create a "long-term memory" a mechanism called time transition function is used to cast previous output in the next input, [Nichele and Gundersen, 2017]. Many types of transition

functions exist however in this dissertation, the normalized addition will be used. The time transition function adds the value of each corresponding cell, if the sum is equal to 2 the state is set to 1, if the sum is equal to 0 the state is set to 0 and if the sum is 1 the function randomly assigns the state 1 or 0. A representation of this mechanism is described in Figure 5.1.

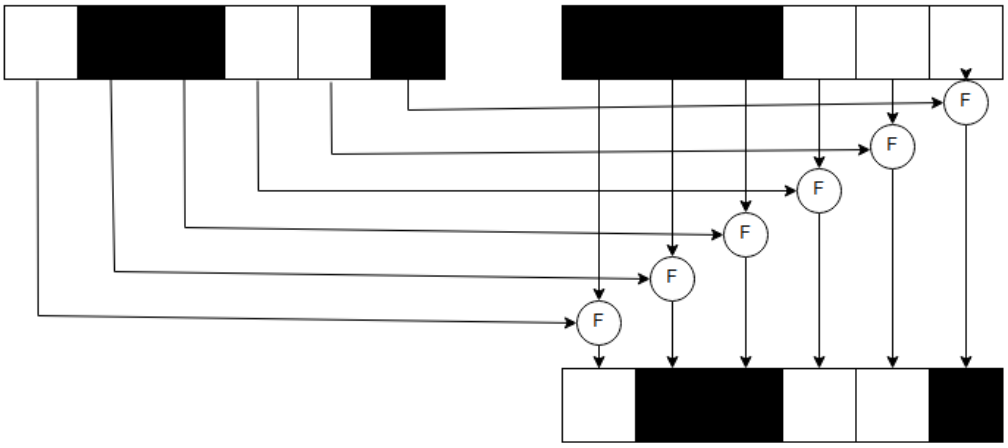

FIGURE 5.1: Time transition function, black cell represents state 1 and white state 0.

- *Input recall*

  To build a "short-term memory" in the system we will need to use a parameter called IR (Input Recall), it defined at which every iteration we will overwrite a small part of the grid states with a sample of the input, [Nichele and Gundersen, 2017]. This way we will allow the system to remember, during the execution, the input data and so create a "short-term memory". This mechanism will force the system into recalling the input multiple times and so evolve it in a way where each input different input should end with a different grid.

Each example of input will be mapped and processed by the reservoir where each iteration will be sent to the readout to train it.

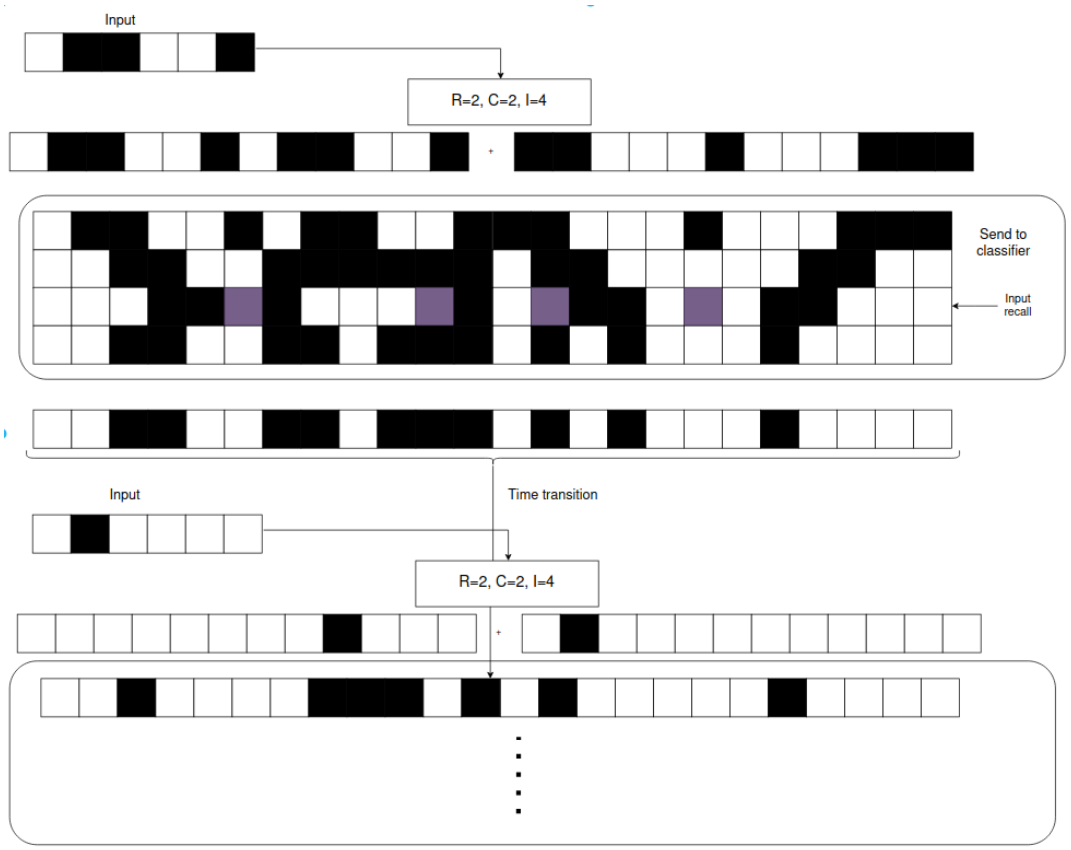

FIGURE 5.2: Partial representation of CA model use as reservoir. A black cell represents state 1 and white state 0, purple indicates data recalled during recall data

5.1.1.3 Rules

We know that 256 rules exist for Binary Elementary cellular automata, a small set of rules has been previously chosen for their computational interest. It has been discovered that the use of multiple rules in the same grid could provide a more refined and nuanced representation of the system dynamics, [Nichele and Gundersen, 2017]. By introducing two rules the CA can capture a higher level of complexity and different patterns otherwise hard to simulate. It has also been proved that two rules that perform well on their own will perform well together, the dual rule mechanism is described in Figure 5.3.

5.1.2 Three states Cellular Automata

Using binary ECA we need to transform our continuous data values into binary this lead to a great amount of data lost. A way to fix this problem is to add more states to our cells in the grid, here we implemented a ECA with three possible states. The encoding stage stays the same with one difference, we now have two thresholds (0.33 and 0.66)

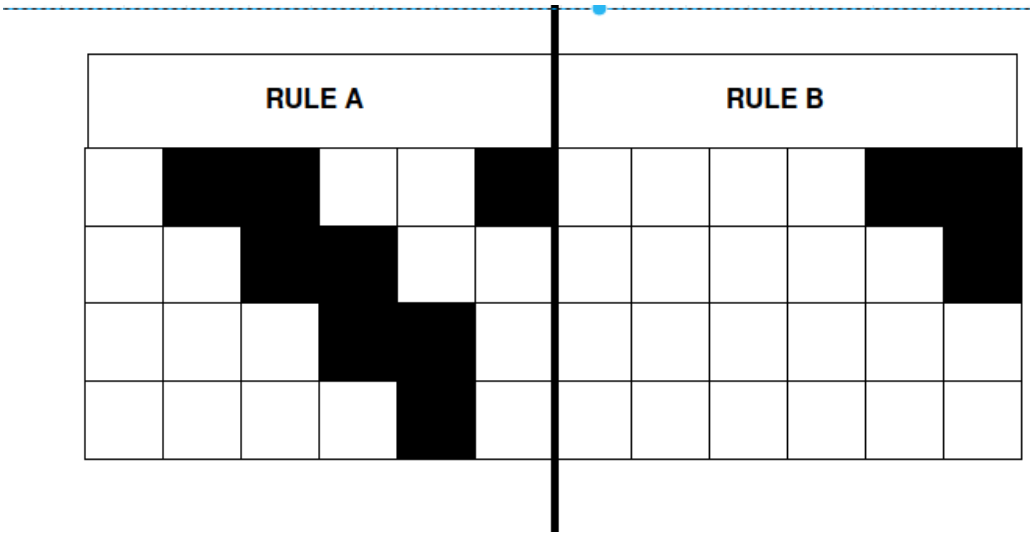

FIGURE 5.3: Description of dual rule ECA, black cell represents state 1 and white state 0.

where each value below the first are set to 0, values between the thresholds are set to 1, and others to 2. The last step is to modify rules so they can perform with three states ECA, we will have to set a state for each 27 three-digit ternary numbers from 000 to 222, as $3^3 = 27$. Then decompose the targeted rule onto a base 3 27-digits number for example rule 33:

$$33 = 1 * 3^3 + 2 * 3^1 - > 000000000000000000000001020$$

This type of CA should allow better performance since the scope of the data is larger than binary ECA and so this will lead to less data lost.

5.1.3 Coupled Map Lattices

In this dissertation, the use of an encoder to modify the data to fit discrete space CA is proposed and we know it leads to data loss. The answer would be to use the same concept but for a continuous space. In answer, we proposed the concept of coupled map lattices, where each cell can take a real number set from a coupling equation. For a "basic" coupling we only consider one neighbor, to compute the state of a site z at t+1 we consider the site itself s at time t and its neighbor site at position s-1.

$$z_n = \epsilon * map(s_n) + (1 - \epsilon) * map(s_{n-1})$$

Here ϵ is called coupling strength it adjusts the degree to which the current state of the sites is taken into consideration. The meaning of the $map()$ function and its utility will be detailed in section 5.1.3.2. Some more complex coupling equations exist one of them is with two neighbors and is defined as followed:

$$z_n = (1 - \epsilon) * map(s_n) + (\epsilon/2) * map(s_{n-1}) + map(s_{n+1})$$

Here the equation takes two neighbors and the site itself and the coupling strength is more or less high for the different neighbors. The encoding aside, replication, and mapping stay the same as with the CA, the global behavior of the system still remains the same as the decoder that sends data to the readout layer.

5.1.3.1 Pinning

In CML the mechanism of pinning refers to the process of fixing the states of certain sites to a specific value in order to force the system into a certain behavior or pattern. Two types of pinning exist, uniform and random, the first one refers to the process of pinning every X site and random means that the system will randomly select X sites to be pinned. Pinning is particularly useful when studying the behavior of complex systems, such as the dynamic of a system or the emergence of specific patterns. The use of both different types of pinning is investigated during the optimization but the uniform pinning shows better results than random pinning.

5.1.3.2 Map

Map in CML represents a rule or equation that governs the evolution of the lattices within the system and shapes the overall evolution of this one. The choice of map function depends on the type of behavior we want to evolve [Lones et al., 2013], three different categories exist, in this dissertation, we will only use discrete and continuous maps. A list of the different maps used is defined as follow: logistic map, sigmoid map, tanh map and circle map. The maps that model non-linear systems are particularly useful as the sigmoid map that has been used as effective low-level elements in recurrent neural networks.

5.1.4 Echo State Network

The purpose of this dissertation is to compare our reservoir to a known one like the ESN, for this purpose we choose to use a Python library called reservoirpy designed by professor Xavier Hinaut, [Trouvain et al., 2020]. This library is a user-friendly tool that allows the user to build reservoir computing network very easily. In this dissertation, we are going to use this library to build a simple echo state network and a Ridge network to be trained, the readout layer will be detailed in section 2.4.1.1. The number of units, the value of the leaking rate and the spectral radius will be determined by the optimization. The leaking rate is a real number between 0 and 1 included that defines how much the previous state will be taken into account when computing the next one, the higher the leaking rate, the less the previous state is taken into account, and the lower the leaking rate, higher is the impact of the previous state in the computation.
Spectral radius defines the maximum absolute eigenvalue of the reservoir in other terms it controls the behavior of the matrix, its stability, and its convergence and so it controls the dynamics of the reservoir when the spectral radius is smaller than 1 the matrix will be more stable and so the dynamics of the reservoir on the other hand a higher spectral radius mean a more chaotic evolution of the matrix and so more chaotics dynamics in the reservoir.

5.2 Optimisation

In this section, we will use a Python library called "sklearn genetic" to optimize our reservoir in order to get the best results we can. Each type of reservoir, CA, 3 states CA, CML and, ESN will be optimized with the same algorithm and the same parameters (the parameters are displayed here 5.1). The algorithm is from a class of genetic algorithms, which means that the optimization search will be based on agents that will evaluate thanks to cross-over or mutation. The big difference and the reason why we will use a genetic algorithm instead of a grid search algorithm is that random search (genetic) will find better sets of hyper-parameters by exploring a larger environment that may be less promising, Bergstra and Bengio [2012]. The computation time will also be smaller, where a genetic algorithm tries to minimize a loss function, and grid search will try every combination of hyper-parameters possible that could take a lot of time to compute.

Parameters	parameter 1	parameter 2	parameter 3	parameter 4	parameter 5	parameter 6	parameter 7
CML	coupling strength	C-parameter	replication	iteration	input recall	map lattices	r
Binary ECA	rule	C-parameter	replication	iteration	input recall	None	None
3 states ECA	rule	C-parameter	replication	iteration	input recall	None	None
ESN	units	leaking rate	spectral radius	None	None	None	None
Readout	leaking rate	None	None	None	None	None	None

TABLE 5.1: This table displays the hyperparameters optimized for each reservoir

This model is also very easy to use, first, we set a parameter space then the algorithm randomly choose a set of combination and fit it to the desired algorithm and compute the score according to the scoring method. From this, a new set of hyper-parameters is defined using mutation and cross-over and once again this set is fitted and scored, the algorithm continues until it reaches the number of generation set. In the end, the combination of hyper-parameter that achieve the best score is fitted and the predictions are made.

Chapter 6

Experimentation and results

To measure the degree of success of a reservoir we will use different metrics, accuracy, computation time and reservoir complexity. The last refers to the size of the reservoir and the rule used, we will test different sizes and combinations of hyper-parameters R, I and C-parameter and analyze the behavior of the reservoir. Computation time is also an interesting metric as a model that performs very well but needs a lot of computation will cost a lot in time and energy. In last the accuracy, we will compute the mean squared error after each execution of the model and measure our accuracy accordingly.
In this section, we will execute our four optimized reservoirs (CA, 3 states CA, CML, ESN) with our three datasets detailed here 2.6, and all metrics and commentary will be based of series of execution and report. Binary ECA will have for each dataset a deeper investigation with not only the best parameters obtained with the optimization but also different combinations of hyper-parameters in order to have a deeper understanding of the model. Each result will be the mean value of twenty executions as one or two executions can be lucky a mean value should be more accurate to the reality of the accuracy.

In section A, for each reservoir and for each experiment, a figure with three graphs has been made. The first graphic represents the data we try to predict, the second represents the same data and the predicted one, and the last graph represents the distance between the reel data and what we have predicted.

6.1 Wind dataset

In this dataset, we try to predict the section called "WIND" which describes the average wind speed. The remaining features describe the temperature, precipitation, and indicators recorded by a meteorological station. Wind speed forecasting is a challenge for meteorologists as it is difficult to predict, this is why it is a good example to test our reservoirs on.

6.1.1 Results

After optimizing our reservoirs, we reported in table 6.1.1 each best-found configuration for CML, Binary ECA, and 3 states ECA and ESN as well as the average mean squared error (rmse) and lowest rmse found in a hundred execution. In order to study the behavior of those reservoirs we entered every rmse in a graphic, 6.1.

Parameters	I or units	R or lr	C-parameter or sr	input recall	coupling strength or rule	time (ms)	Average rmse	Lowest rmse
CML	8	3	2	3	0.64	10216.76	5.98	4.96
Binary ECA	8	4	4	3	90:60	17801.53	5	4.98
3 states ECA	5	4	3	1	110	3851.02	8.73	5.89
ESN	477	0.902	1.86	None	None	888.98	33.90	9.34

The first thing we can see in figure 6.1 is that ESN has very unstable results, on the other hand, the other reservoirs are very stable and the resulting space stays small. The parameters I, R, and C-parameter showed in table 6.1.1, were, during the optimization caped respectively to 8, 4, and 4 so we can conclude that reservoirs seem to get better accuracy with bigger grid size. At last, the input recall was caped to 3 it seems that the nature of the discrete dynamical system itself impacts this parameter as we see here the maximum and minimum values. With this dataset, the best reservoir tends to be the Binary CA and the three states CA.

In table, 6.1 we reported the average rmse of ten executions of every combination of rules tried, the goal is to analyze the different combinations of rules and isolate the best one and see if rules that perform well alone also perform well together. We found here

rule	60	90	102	105	150	153	165	195	210
60	5.04	4.91	5.09	5.04	5.03	4.99	4.91	4.94	5.03
90	5.06	5.08	5.08	4.90	4.90	5.02	4.81	5.01	5.01
102	5.06	4.87	5.01	4.90	5.02	4.96	4.87	4.97	5.02
105	4.97	4.94	4.95	4.91	5.03	4.94	4.97	4.94	5.02
150	5.01	4.92	5.00	4.92	5.00	4.90	4.89	4.98	4.98
153	4.95	4.90	5.03	4.99	4.96	5.04	5.04	4.91	4.93
165	4.98	4.94	4.92	5.07	5.03	5.01	4.91	4.99	4.96
195	5.02	4.98	4.99	4.94	4.99	4.88	4.98	5.08	4.95
210	5.05	4.98	5.08	4.98	5.06	4.99	5.08	4.97	5.04

TABLE 6.1: Binary ECA with all combinations of rules tested 20 times. Each result is the average rmse. Results in red are used for better reading as they represent pair of rules.

that this is partially true as sometimes rules that perform in the average can perform well together.

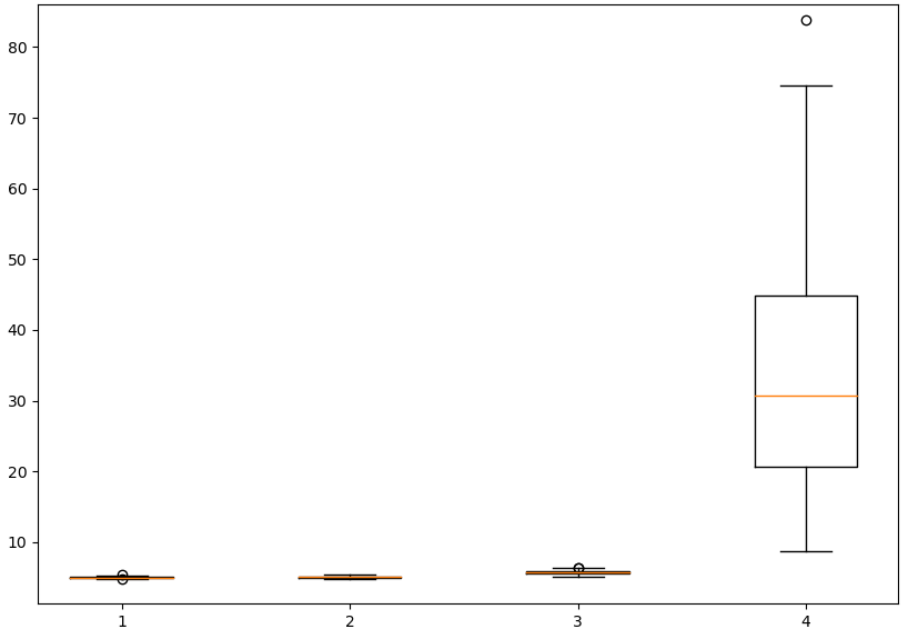

FIGURE 6.1: Box plot of 100 execution representing the mean squared error. 1: CML, 2: Binary ECA, 3:ECA 3 states, 4:ESN.

In the end, for a better visualisation and understanding we can see on figure 6.2 a simple execution of one reservoir on the Wind dataset. This figure shows our predicted data on the targeted data, this way we visualise how our model fit this one. The last graphic on the figure is the targeted feature minus our prediction, this way we can quickly identify where the model is the most wrong. We see that our model fit the general behavior

of the data with some errors but the radius of the predicted values seems in the same radius as the targeted data. In section A a figure for each reservoir on this dataset is available from figure A.1 to A.4.

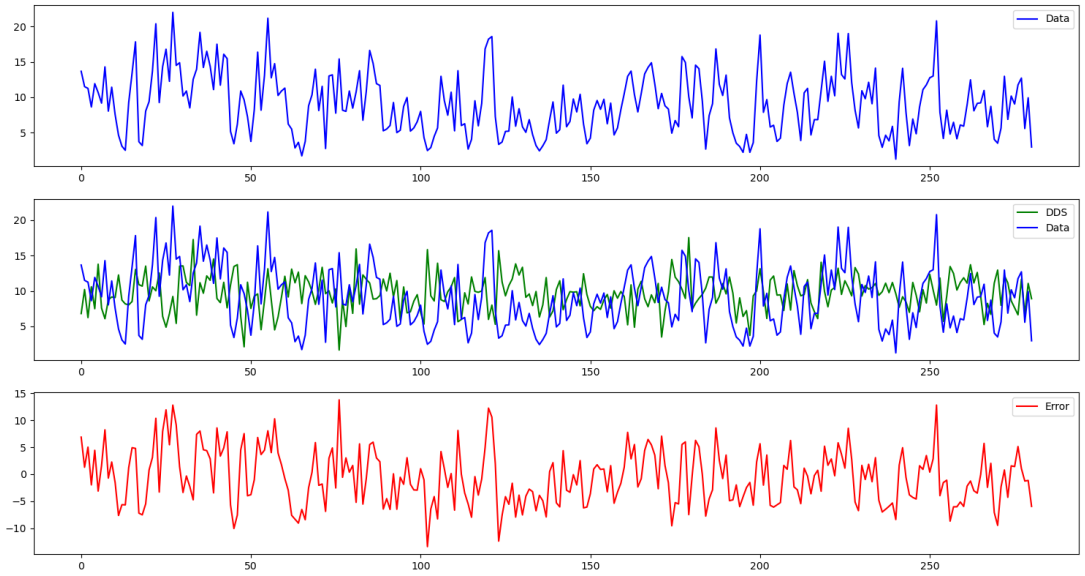

FIGURE 6.2: Binary ECA predictions on targeted data and error curve (*target* − *prediction*) Wind dataset

6.2 DIJA dataset

This dataset describes the daily news stock market for Dow Jones Industrial Average, an American company. We will try to predict the feature called "Open" which describes the open price for the day, the remaining features are about other pricing and volume. Forecasting stock market data is another high challenge, the system that the stock market describe is highly chaotic and hard to predict this is a fitting task for our reservoirs.

6.2.1 Results

In the case of the DIJA dataset, we reported our best configuration for reservoirs CML, Binary ECA, three states ECA and ESN in table 6.2.1. By comparing table 6.2.1 and table 6.1.1 we see that the hyper-parameters are the same apart few small changes. CML and Binary ECA seem to get better results with a greater number of iterations, and a greater number of R and C-parameter. Here the coupling strength is quite the

same but the rule used in Binary ECA and three-state ECA changed, we can theorize that for different experimentation a specific rule and so the behavior of the reservoir is required, one efficient rule could not be this efficient in another experimentation with another dataset. In figure 6.5 a very large sample of different result for ESN strengthen the idea that a neural-based reservoir shows unpredictable results and a large number of execution is required to obtain good results. But, ESN is, in this case, the reservoir with the best accuracy on a single "best" execution.

Parameters	I or units	R or lr	C-parameter or sr	input recall	coupling strength or rule	time (ms)	Average rmse	Lowest rmse
CML	8	2	3	3	0.60	3771.62	7409.45	4368.45
Binary ECA	8	4	4	3	102:153	6511.06	5456.34	4272.23
3 states ECA	5	4	3	2	30	1148.97	5302.34	5003.45
ESN	303	0.086	1.29	None	None	408.80	7545.34	3587.34

Table 6.2 demonstrates every combination of rules we tried, the average rmse of 4272.23 is shown multiple times for the row of rule 60, this is due to the rule itself that terminates every set of iterations with every cell at state zero and the combined rule cannot always overwrite this behavior. In conclusion, this number refers to the grid in the last iteration with all cells at state 0, and so the readout layer is unable to train and the resulting output is always equal to the first case encountered. But some rules can on their side of the grid demonstrate an evolution of the system and achieve great accuracy like rules 102 and 60 or rule 153 and 60. Maybe adding some cells with state zero at both sides of the ECA could improve the accuracy of the reservoir.

In the end, for a better visualisation and understanding we can see on figure 6.4 a simple execution of one reservoir on the DIJA dataset. This figure shows our predicted data on the targeted data, this way we visualise how our model fit this one. The last graphic on the figure is the targeted feature minus our prediction, this way we can quickly identify where the model is the most wrong. Here, our model seems to be very unstable with our predictions, the radius of our predicted values is much greater than the targeted values but we know that stock market is a very hard system to predict and this revoir shows in a way good results. In section A a figure for each reservoir on this dataset is available

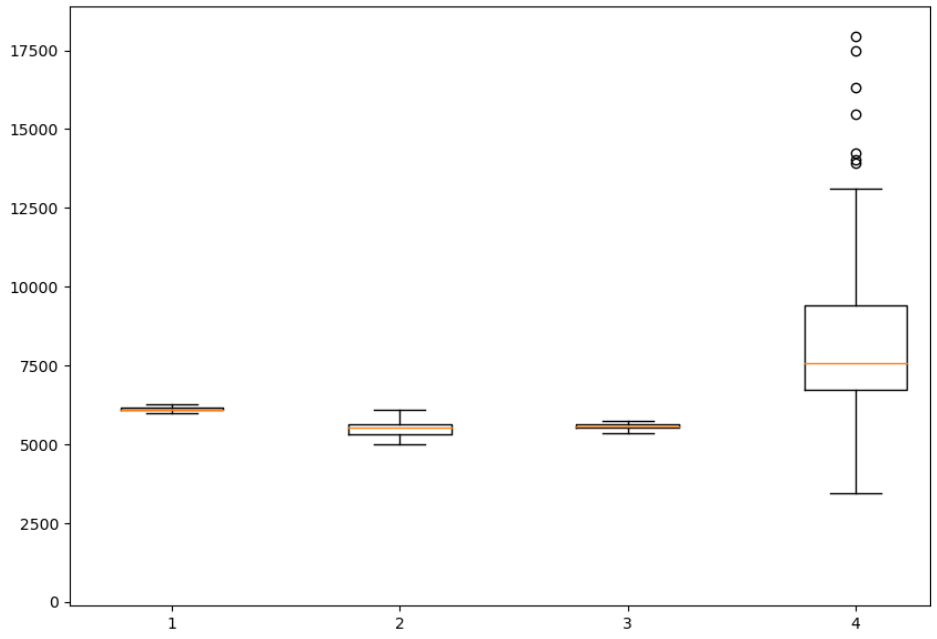

FIGURE 6.3: Box plot of 100 execution representing the mean squared error. 1: CML, 2: Binary ECA, 3:ECA 3 states, 4:ESN.

rule	60	90	102	105	150	153	165	195	210
60	4272.23	4272.23	4272.23	5724.76	4272.23	5811.37	5705.71	5805.03	4272.23
90	5417.86	5699.11	5428.87	5528.99	5549.56	5601.93	5474.35	5436.86	5415.12
102	5377.49	5506.68	5469.90	5522.95	5677.77	5416.40	5375.73	5365.74	5394.60
105	5411.61	5578.56	5490.21	5498.74	5399.53	5496.05	5502.46	5439.21	5494.53
150	5572.66	5388.07	5566.70	5450.97	5377.13	5458.23	5351.02	5532.26	5617.49
153	5328.44	5579.99	5547.19	5520.89	5447.20	5554.71	5552.38	5161.21	5554.43
165	5467.36	5485.50	5512.17	5587.14	5547.64	5554.11	5547.83	5547.05	5752.77
195	5235.91	5484.84	5505.83	5517.46	5568.05	5510.32	5481.85	5554.62	5335.45
210	5459.13	5453.93	5650.90	5450.03	5747.93	5429.88	5409.24	5497.44	5469.68

TABLE 6.2: Binary ECA with all combinations of rules tested 20 times. Each result is the average rmse. Results in red are used for better reading as they represent pair of rules.

from figure A.5 to A.8.

6.3 Sunspots dataset

This last set of data is about a natural phenomenon called sunspots, here we have two features the mean number of sunspots for a month and the date. In order to get a target feature to predict and features to help us forecast we changed the dataset and created

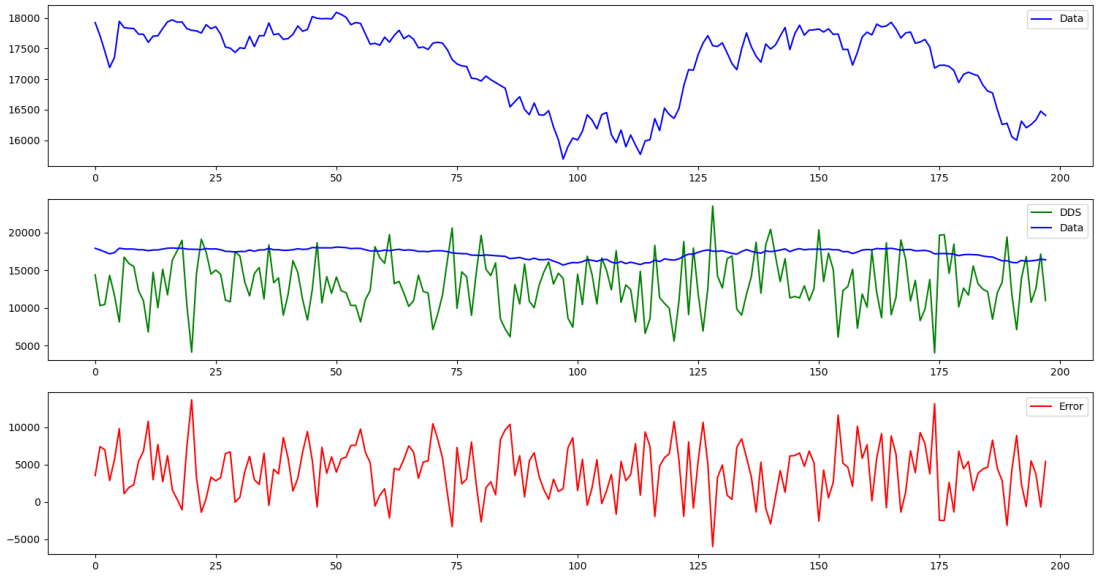

FIGURE 6.4: Binary ECA predictions on targeted data and error curve (*target* − *prediction*) DIJA dataset

a column with the mean number of sunspots for a period of 4 months that will be our target feature and the mean number of sunspots for every month in this period. As this natural phenomenon is still not entirely understand and hard to study it is a good chaotic system to test our reservoirs on.

6.3.1 Results

As we explained in section 6.3 this dataset has to be tuned a bit as it only contained only one feature but this should not affect our results or experiment. Table 6.3.1 strengthens the idea that a large grid and so great number of parameters I, R are needed, C-parameter seems to be in the case of CML to not be needed as great as in ECA, this can be explained because setting a cell to zero in a system where the next generation of cells' states are determined by an equation involving current state could lead to the results equal to zero and so decreasing the accuracy. In the experimentation, the coupling strength is again quite the same as before and the rule used for Binary ECA and three ECA are the same as for the wind dataset, this shows us that maybe they are rules that can be efficient for different problems and so a dynamic system can solve multiple problems with a same or close combination of hyper-parameter.

Parameters	I or units	R or lr	C-parameter or sr	input recall	coupling strength or rule	time	Average rmse	Lowest rmse
CML	8	3	2	3	0.64	4341.90	56.34	50.79
Binary ECA	8	4	4	3	90:60	10330.80	74.03	55.39
3 states ECA	5	4	3	1	110	1999.87	62.35	59.45
ESN	128	0.025	1.25	None	None	590.35	75.34	42.34

Figure 6.5 shows that ESN has again a large different number of results, but achieves again the best accuracy, CML also has good results and the smallest space of results where Binary ECA and three states ECA are not as good and have large space of results. Table 6.3 shows us that rules that perform well alone like 195 or 60 perform well together as they achieved together a small rmse of 55.39.

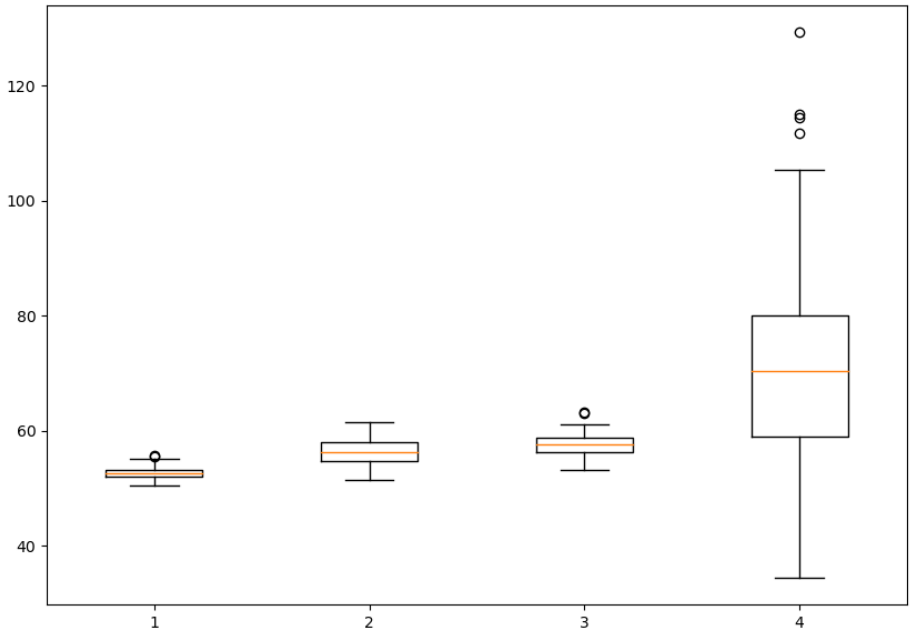

FIGURE 6.5: Box plot of 100 execution representing the mean squared error. 1: CML, 2: Binary ECA, 3:ECA 3 states, 4:ESN.

In the end, for a better visualisation and understanding we can see on figure 6.6 a simple execution of one reservoir on the Sunspots dataset. This figure shows our predicted data on the targeted data, this way we visualise how our model fit this one. The last graphic

rule	60	90	102	105	150	153	165	195	210
60	70.27	69.59	68.90	59.43	66.03	60.44	60.43	59.56	72.50
90	65.28	66.02	66.23	60.68	60.22	64.40	64.33	65.07	66.49
102	115.61	105.95	142.97	75.12	82.42	75.33	76.54	76.08	100.13
105	77.75	75.83	76.63	81.54	77.05	75.11	75.63	75.50	76.80
150	76.09	73.53	74.95	73.21	74.31	74.37	74.61	74.11	75.70
153	73.28	74.62	74.34	73.51	73.18	75.32	74.51	73.79	75.31
165	55.43	62.75	59.10	67.08	65.20	71.62	72.89	72.37	61.23
195	55.39	62.06	55.74	63.34	57.60	71.37	71.81	71.24	59.24
210	74.61	73.83	75.48	73.65	72.88	74.74	74.76	73.71	73.62

TABLE 6.3: Binary ECA with all combinations of rules tested 20 times. Each result is the average rmse. Results in red are used for better reading as they represent pair of rules.

on the figure is the targeted feature minus our prediction, this way we can quickly identify where the model is the most wrong. This datset is where our model are the most accurate, we see here that our predictions fit close to exactly the behavior of the targeted data, our predictions seem to be a little bit higher than expected but a finer tuning of the hyperparameters could fix this error. In section A a figure for each reservoir on this dataset is available from figure A.9 to A.12.

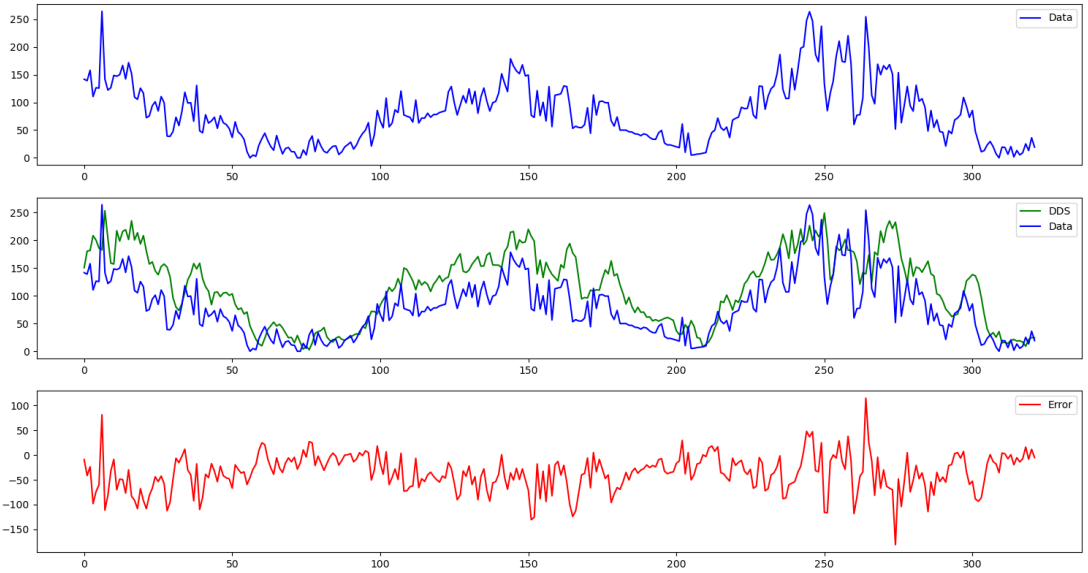

FIGURE 6.6: ESN predictions on targeted data and error curve ($target - prediction$) Sunspots dataset

Parameters	Wind dataset	DIJA dataset	Sunspots dataset
CML	4.96	4368.45	50.79
Binary ECA	4.98	4272.23	55.39
3 states ECA	5.89	5003.45	59.45
ESN	9.34	3587.34	42.34

TABLE 6.4: Overall table of each reservoir's best rmse on each dataset

6.4 Overall

In the end, we can conclude that DDS as a reservoir achieves better stability than ESN which needs to be executed multiple times in order to find a fair accuracy. We also found that DDS are more robust than ESN in the way that the same configuration of parameters for different experiments can lead to good accuracy for both.

ESN seems to achieve better accuracy than ESN in a smaller amount of time but DDS seems to be a more reliable and simpler system to use. A table recording the best execution of each reservoir on each dataset has been displayed here 6.4.

Chapter 7

Conclusion and future work

7.1 Conclusion

In conclusion, we understand that the use of DDS within a recurrent neural network shows to be an optimistic more reliable, and simpler model. In terms of accuracy, those models are not far from the ESN accuracy and can achieve it in a quicker time, they can also be more robust face to different problems as one combination of rules can solve multiple problems and a large number of rules are available and so DDS could solve a large number of problems. As the ESN is a neural network it is not an easy thing to understand how we achieve our output through the neurons, on the contrary, DDS are really easy to explain and use threw simple rules. The choice between ESN and the use of DDS as a reservoir depends on our needs and requirements.
We would say that this work answer all previous goals and know help understand the different strength of different types of reservoir. If we had to start this project again we think would add more dataset to the experimentation section and maybe use better hardware to enhance the optimization section.

7.2 Future Work

The results presented in this dissertation show that a discrete dynamical system as a reservoir performs in certain ways better than a neural reservoir. This dissertation briefly investigates this large spectrum that is discrete dynamical systems as reservoirs.

A deeper optimization and analysis could show even better results and advantages of dynamical systems.

We investigated different datasets but all of them showed reel values, because of the binary nature of ECA a discrete dynamical system with a discrete space, ESN has the advantage of needing no transformation of the data, to analyze the different metrics of these two reservoirs we could also use a binary dataset.

One of the most interesting future works will be to re-investigate dynamical systems as reservoirs but with multiple reservoirs, a mechanism called deep reservoir system.

Another interesting part to be worked on is the mapping function, indeed, we use in this dissertation a time transition function but a lot of mapping function exists and could improve the performance of the reservoir.

Appendix A

Graphic predictions and error

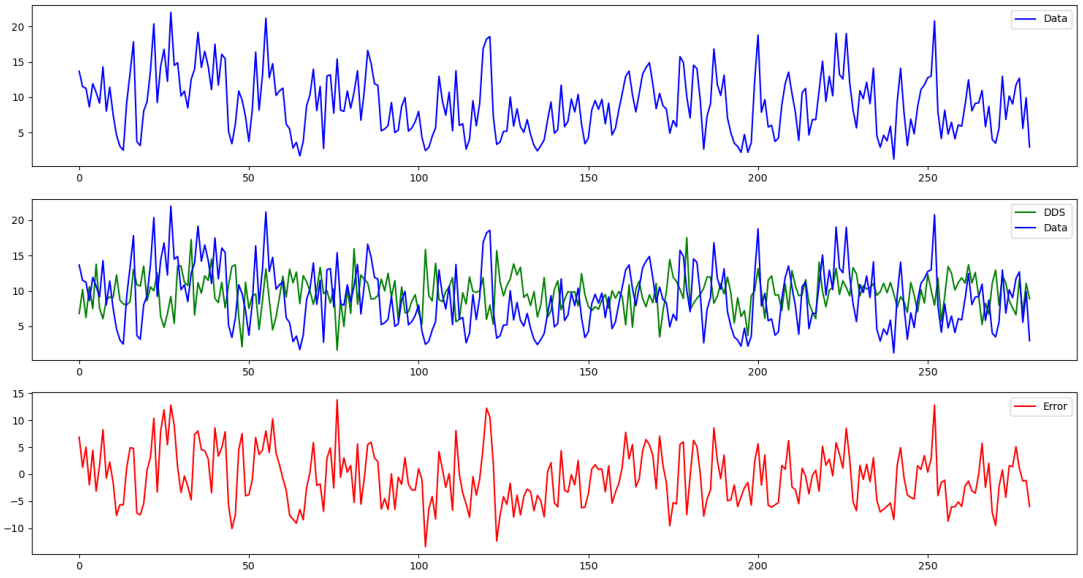

FIGURE A.1: Binary ECA with Wind dataset.

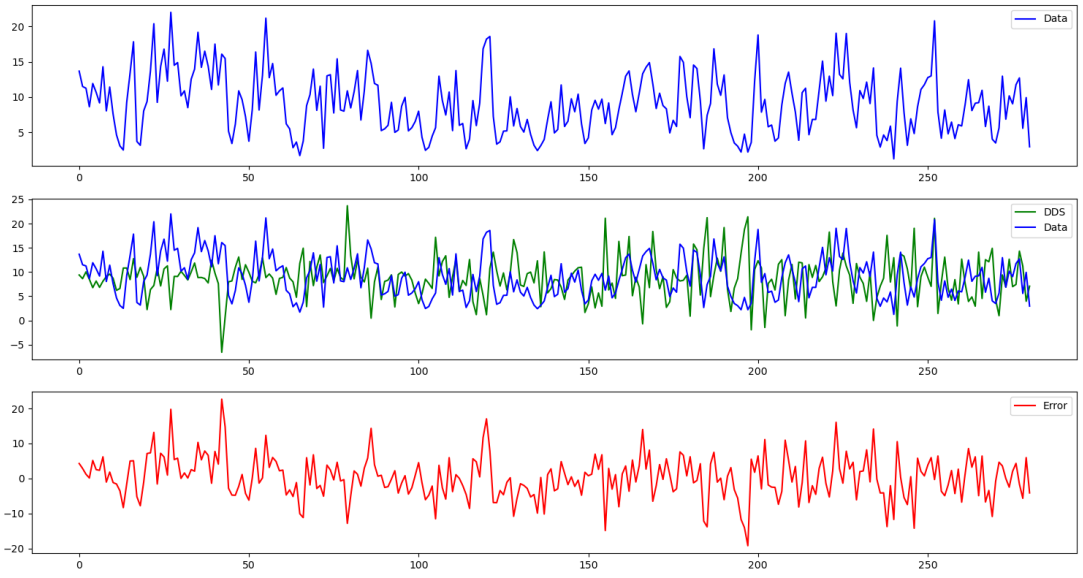

FIGURE A.2: Three states ECA with Wind dataset.

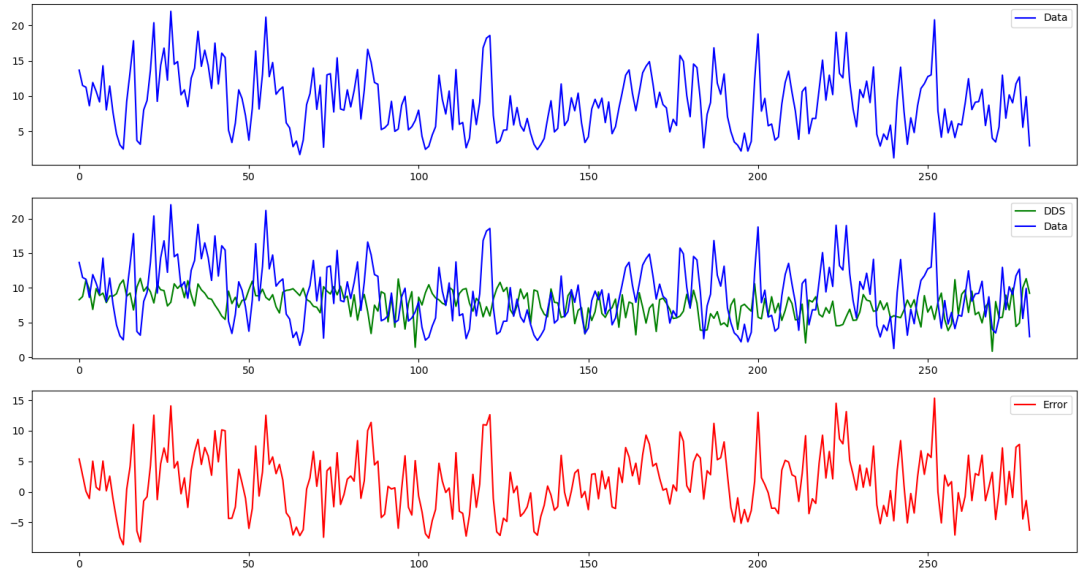

FIGURE A.3: CML with Wind dataset.

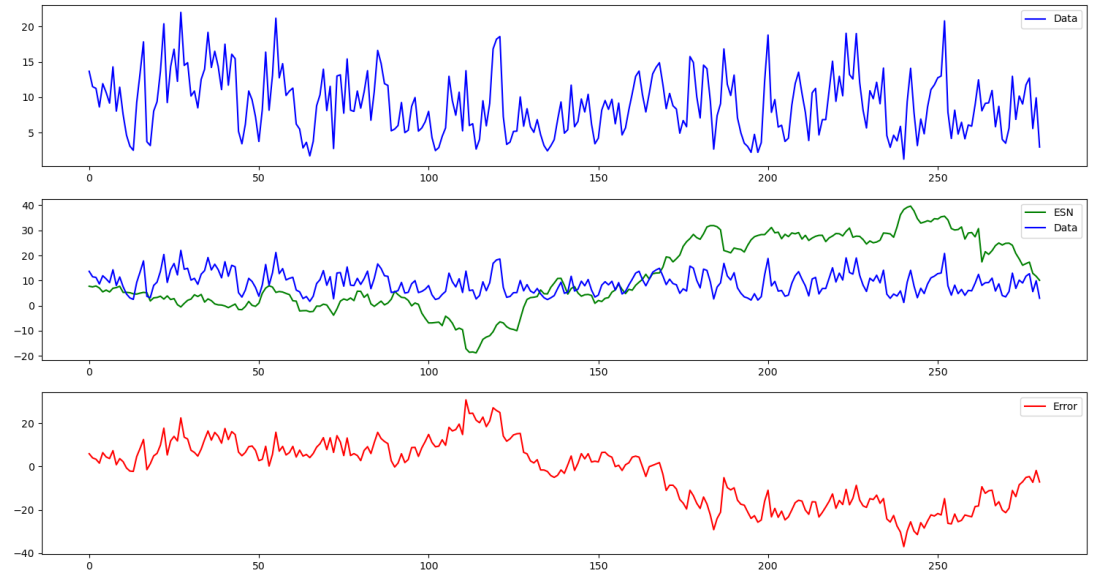

FIGURE A.4: ESN with Wind dataset.

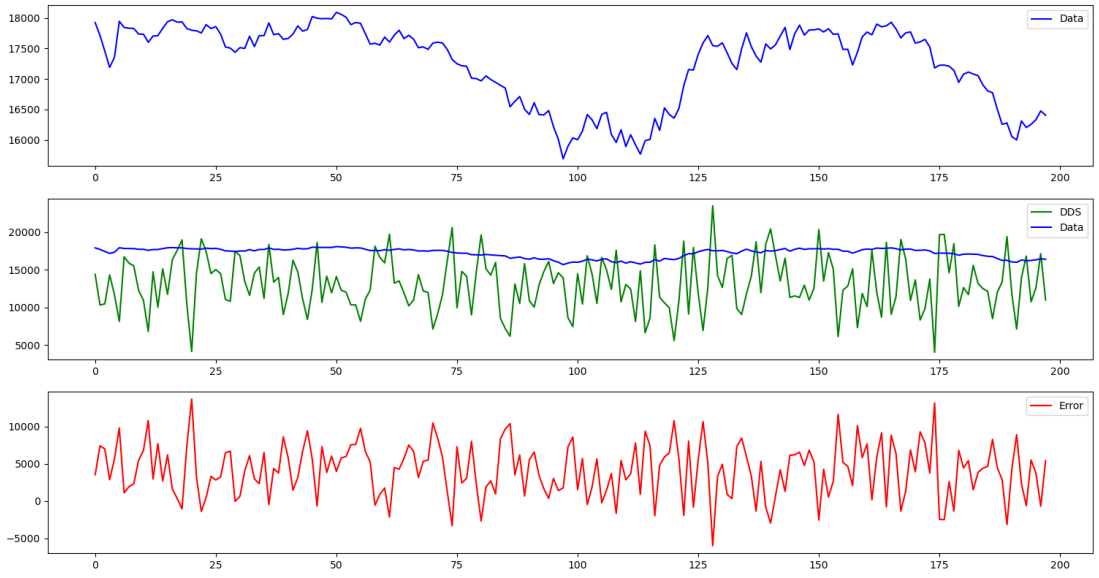

FIGURE A.5: Binary ECA with DIJA dataset.

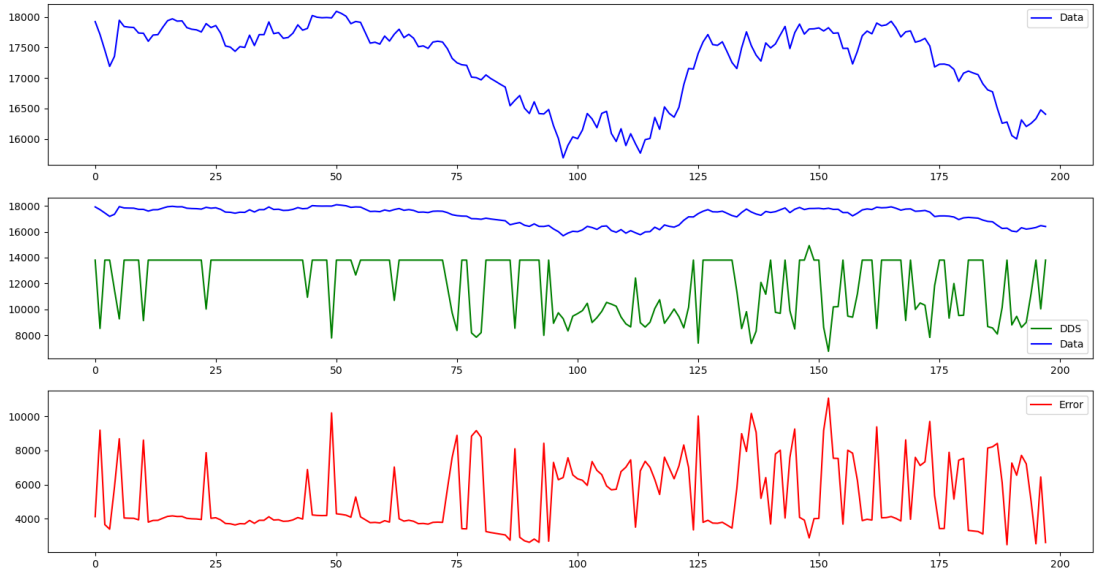

FIGURE A.6: Three states ECA with DIJA dataset.

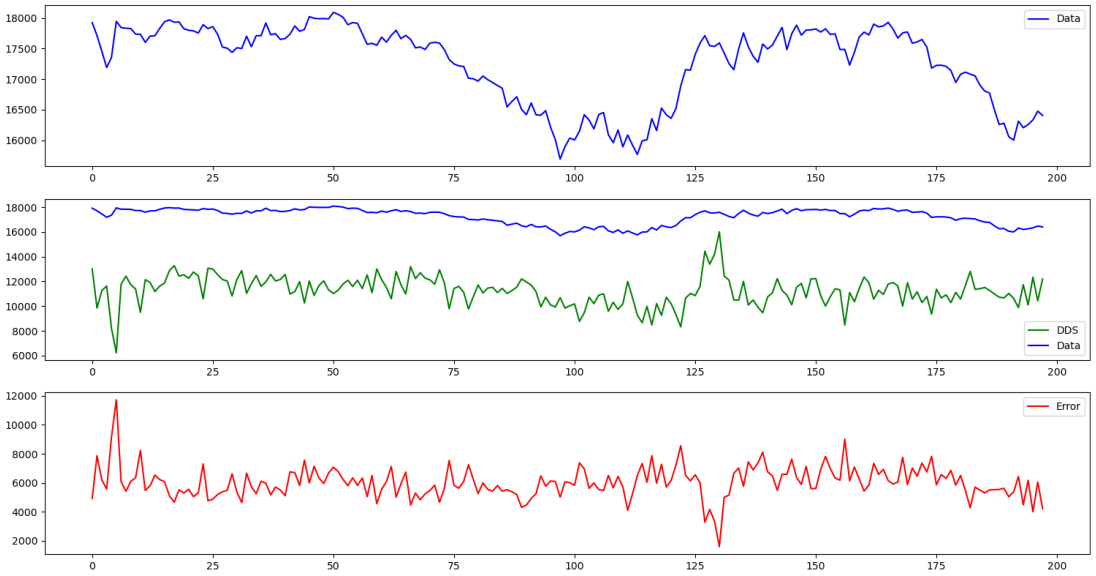

FIGURE A.7: CML with DIJA dataset.

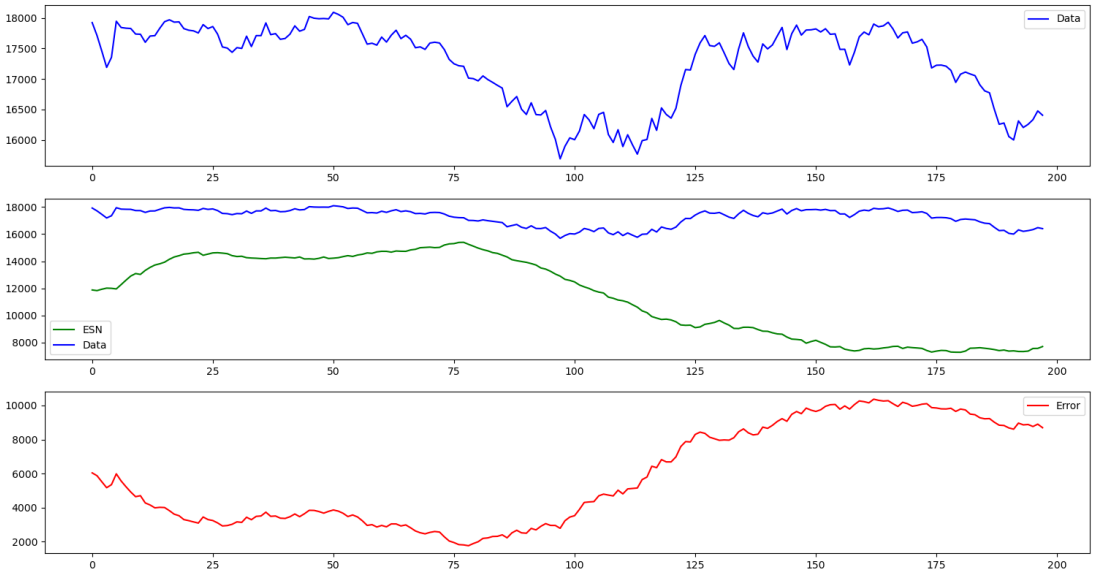

FIGURE A.8: ESN with DIJA dataset.

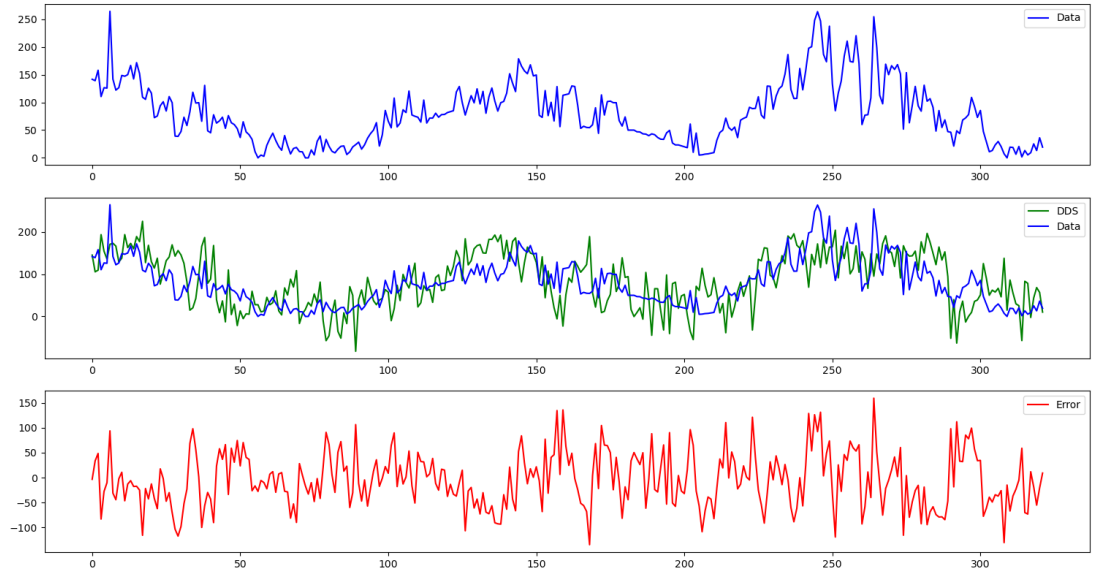

FIGURE A.9: Binary ECA with Sunspots dataset.

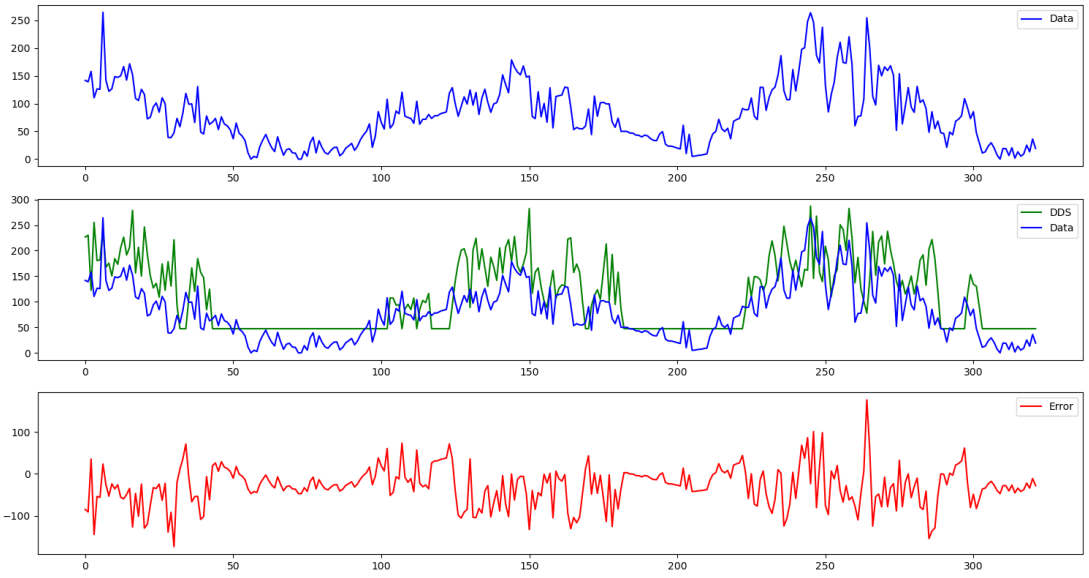

FIGURE A.10: Three states ECA with Sunspots dataset.

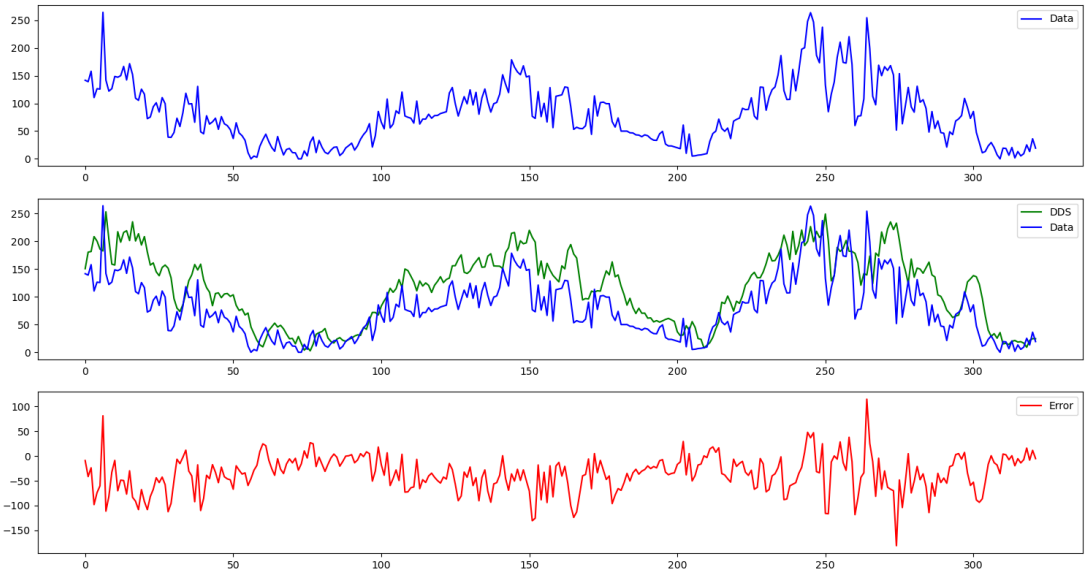

FIGURE A.11: CML with Sunspots dataset.

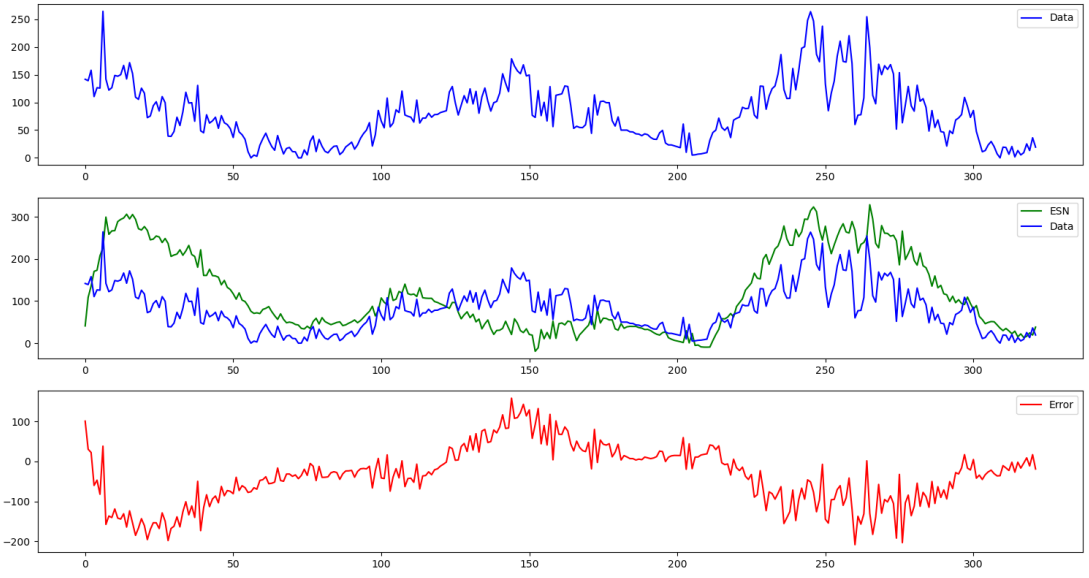

FIGURE A.12: ESN with Sunspots dataset.

Bibliography

Babson, N. and Teuscher, C. [2019], 'Reservoir computing with complex cellular automata.', *Complex Systems* **28**(4).

Bergstra, J. and Bengio, Y. [2012], 'Random search for hyper-parameter optimization.', *Journal of machine learning research* **13**(2).

Cisneros, H., Mikolov, T. and Sivic, J. [2022], Benchmarking learning efficiency in deep reservoir computing, *in* S. Chandar, R. Pascanu and D. Precup, eds, 'Proceedings of The 1st Conference on Lifelong Learning Agents', Vol. 199 of *Proceedings of Machine Learning Research*, PMLR, pp. 532–547.
URL: *https://proceedings.mlr.press/v199/cisneros22a.html*

Clegg, D. and Barker, R. [1994], *Case method fast-track: a RAD approach*, Addison-Wesley Longman Publishing Co., Inc.

Crooks, A. [2017], 'Cellular automata', *The International Encyclopedia of Geography: People, the Earth, Environment, and Technology* pp. 1–9.

Gallicchio, C., Micheli, A. and Pedrelli, L. [2018], 'Design of deep echo state networks', *Neural Networks* **108**, 33–47.
URL: *https://www.sciencedirect.com/science/article/pii/S0893608018302223*

Gardner, M. [1970], 'The fantastic combinations of jhon conway's new solitaire game'life', *Sc. Am.* **223**, 20–123.

Grebogi, C., Ott, E. and Yorke, J. A. [1982], 'Chaotic attractors in crisis', *Physical Review Letters* **48**(22), 1507.

Grössing, G. and Zeilinger, A. [1988], 'Quantum cellular automata', *Complex systems* **2**(2), 197–208.

Ilachinski, A. [2001], *Cellular automata: a discrete universe*, World Scientific Publishing Company, pp. 12–13.

Jaeger, H. and Haas, H. [2004], 'Harnessing nonlinearity: Predicting chaotic systems and saving energy in wireless communication', *science* **304**(5667), 78–80.

Kaneko, K. [1992], 'Overview of coupled map lattices', *Chaos: An Interdisciplinary Journal of Nonlinear Science* **2**(3), 279–282.

Lones, M. A., Fuente, L. A., Turner, A. P., Caves, L. S., Stepney, S., Smith, S. L. and Tyrrell, A. M. [2013], 'Artificial biochemical networks: Evolving dynamical systems to control dynamical systems', *IEEE transactions on evolutionary computation* **18**(2), 145–166.

Maass, W. [2011], 'Liquid state machines: motivation, theory, and applications', *Computability in context: computation and logic in the real world* pp. 275–296.

McCulloch, W. S. and Pitts, W. [1943], 'A logical calculus of the ideas immanent in nervous activity', *The bulletin of mathematical biophysics* **5**, 115–133.

Nichele, S. and Gundersen, M. S. [2017], 'Reservoir computing using non-uniform binary cellular automata', *arXiv preprint arXiv:1702.03812* .

Nichele, S. and Molund, A. [2017*a*], 'Deep learning with cellular automaton-based reservoir computing', *Complex Systems* **26**.

Nichele, S. and Molund, A. [2017*b*], 'Deep learning with cellular automaton-based reservoir computing'.

Qi, A.-S., Zheng, X., Du, C.-Y. and An, B.-S. [1993], 'A cellular automaton model of cancerous growth', *Journal of theoretical biology* **161**(1), 1–12.

Rendell, P. [2002], 'Turing universality of the game of life', *Collision-based computing* pp. 513–539.

Rosenblatt, F. [1958], 'The perceptron: a probabilistic model for information storage and organization in the brain.', *Psychological review* **65**(6), 386.

Rumelhart, D. E., Hinton, G. E. and Williams, R. J. [1986], 'Learning representations by back-propagating errors', *nature* **323**(6088), 533–536.

Schiff, J. L. [2011], *Cellular automata: a discrete view of the world*, John Wiley & Sons.

Szita, I., Gyenes, V. and Lőrincz, A. [2006], Reinforcement learning with echo state networks, *in* 'Artificial Neural Networks–ICANN 2006: 16th International Conference, Athens, Greece, September 10-14, 2006. Proceedings, Part I 16', Springer, pp. 830–839.

Trouvain, N., Pedrelli, L., Dinh, T. T. and Hinaut, X. [2020], Reservoirpy: an efficient and user-friendly library to design echo state networks, *in* 'International Conference on Artificial Neural Networks', Springer, pp. 494–505.

Uguz, S. and Redjepov, S. [2021], Reflexive and adiabatic boundary 2d linear cellular automata and evolution of image patterns, *in* '2021 International Conference on Information Science and Communications Technologies (ICISCT)', IEEE, pp. 1–7.

Yilmaz, O. [2014], 'Reservoir computing using cellular automata', *arXiv preprint arXiv:1410.0162* .